(51) International Patent Classification[7]: G06F 15/16

(21) International Application Number: PCT/US00/26728

(22) International Filing Date:
29 September 2000 (29.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/159,086    13 October 1999 (13.10.1999)    US
09/672,909    28 September 2000 (28.09.2000)    US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:
US                              60/159,086 (CIP)
Filed on          13 October 1999 (13.10.1999)
US                              09/672,709 (CIP)
Filed on          28 September 2000 (28.09.2000)

(71) Applicant (for all designated States except US): TIMES N SYSTEMS, INC. [US/US]; Bldg. B, Suite P, 1908 Kramer Lane, Austin, TX 78758 (US).

(72) Inventor; and
(75) Inventor/Applicant (for US only): BRIDGERS, Vince

[US/US]; 11337 Pebble Garden Lane, Austin, TX 78739 (US).

(74) Agent: BRUCKNER, John, J.; Wilson Sonsini Goodrich & Rosati, 650 Page Mill Road, Palo Alto, CA 94304-1050 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— Without international search report and to be republished upon receipt of that report.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: LOW LATENCY, HIGH BANDWIDTH MULTI-COMPUTER SYSTEM INTERCONNECT

(57) Abstract: Methods, systems and devices are described for a low latency, high bandwidth multi-computer system interconnect. A method includes passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including: receiving said set of interconnect fabric data with said shim layer, classifying said set of interconnect fabric data with said shim layer, and handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated. The methods, systems and devices provide advantages because the speed and scalability of parallel processor systems is enhanced.

# LOW LATENCY, HIGH BANDWIDTH MULTI-COMPUTER SYSTEM INTERCONNECT

## REFERENCE TO APPENDIX

An appendix is included in this application by way of attachment, the totality of which is hereby incorporated by reference as an integral part of this application. The appendix includes printed source code that is discussed below in more detail as a nonlimiting example of the invention.

## BACKGROUND OF THE INVENTION

### 1.    Field of the Invention

The invention relates generally to the field of computer systems which have multiple processing nodes and in which each processing node is provided with private, local memory and also in which each processing node has access to a range of memory which is shared with other processing nodes. More particularly, the invention relates to computer science techniques that utilize a low latency, high bandwidth multi-computer system interconnect.

### 2.    Discussion of the Related Art

The clustering of workstations is a well-known art. In the most common cases, the clustering involves workstations that operate almost totally independently, utilizing the network only to share such services as a printer, license-limited applications, or shared files.

In more-closely-coupled environments, some software packages (such as NQS) allow a cluster of workstations to share work. In such cases the work arrives, typically as batch jobs, at an entry point to the cluster where it is queued and dispatched to the workstations on the basis of load.

In both of these cases, and all other known cases of clustering, the operating system and cluster subsystem are built around the concept of message-passing. The term message-passing means that a given workstation operates on some portion of a job until communications (to send or receive data, typically) with another workstation is necessary. Then, the first workstation

prepares and communicates with the other workstation.

Another well-known art is that of clustering processors within a machine, usually called a Massively Parallel Processor or MPP, in which the techniques are essentially identical to those of clustered workstations. Usually, the

5    bandwidth and latency of the interconnect network of an MPP are more highly optimized, but the system operation is the same.

In the general case, the passing of a message is an extremely expensive operation; expensive in the sense that many CPU cycles in the sender and receiver are consumed by the process of sending, receiving, bracketing,

10   verifying, and routing the message, CPU cycles that are therefore not available for other operations. A highly streamlined message-passing subsystem can typically require 10,000 to 20,000 CPU cycles or more.

There are specific cases wherein the passing of a message requires significantly less overhead. However, none of these specific cases is adaptable

15   to a general-purpose computer system.

Message-passing parallel processor systems have been offered commercially for years but have failed to capture significant market share because of poor performance and difficulty of programming for typical parallel applications. Message-passing parallel processor systems do have some

20   advantages. In particular, because they share no resources, message-passing parallel processor systems are easier to provide with high-availability features. What is needed is a better approach to parallel processor systems.

There are alternatives to the passing of messages for closely-coupled cluster work. One such alternative is the use of shared memory for inter-

25   processor communication.

Shared-memory systems, have been much more successful at capturing market share than message-passing systems because of the dramatically superior performance of shared-memory systems, up to about four-processor systems. In Search of Clusters, Gregory F. Pfister 2nd ed. (January 1998) Prentice Hall

30   Computer Books, ISBN: 0138997098 describes a computing system with multiple processing nodes in which each processing node is provided with private, local memory and also has access to a range of memory which is shared with other processing nodes. The disclosure of this publication in its entirety is

hereby expressly incorporated herein by reference for the purpose of indicating the background of the invention and illustrating the state of the art.

However, providing high availability for traditional shared-memory systems has proved to be an elusive goal. The nature of these systems, which

5 share all code and all data, including that data which controls the shared operating systems, is incompatible with the separation normally required for high availability. What is needed is an approach to shared-memory systems that improves availability.

Although the use of shared memory for inter-processor communication

10 is a well-known art, prior to the teachings of U.S. Ser. No. 09/273,430, filed March 19, 1999, entitled Shared Memory Apparatus and Method for Multiprocessing Systems, the processors shared a single copy of the operating system. The problem with such systems is that they cannot be efficiently scaled beyond four to eight way systems except in unusual circumstances. All known

15 cases of said unusual circumstances are such that the systems are not good price-performance systems for general-purpose computing.

The entire contents of U.S. Patent Applications 09/273,430, filed March 19, 1999 and PCT/US00/01262, filed January 18, 2000 are hereby expressly incorporated by reference herein for all purposes. U.S. Ser. No. 09/273,430,

20 improved upon the concept of shared memory by teaching the concept which will herein be referred to as a tight cluster. The concept of a tight cluster is that of individual computers, each with its own CPU(s), memory, I/O, and operating system, but for which collection of computers there is a portion of memory which is shared by all the computers and via which they can exchange

25 information. U.S. Ser. No. 09/273,430 describes a system in which each processing node is provided with its own private copy of an operating system and in which the connection to shared memory is via a standard bus. The advantage of a tight cluster in comparison to an SMP is "scalability" which means that a much larger number of computers can be attached together via a

30 tight cluster than an SMP with little loss of processing efficiency.

What is needed are improvements to the concept of the tight cluster. What is also needed is an expansion of the concept of the tight cluster.

## SUMMARY OF THE INVENTION

A goal of the invention is to simultaneously satisfy the above-discussed requirements of improving and expanding the tight cluster concept which, in the case of the prior art, are not satisfied.

One embodiment of the invention is based on a method comprising: passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including: receiving said set of interconnect fabric data with said shim layer, classifying said set of interconnect fabric data with said shim layer, and handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated. Another embodiment of the invention is based on an apparatus, comprising: a shared memory unit; a first system coupled to said shared memory unit; and a second system coupled to said shared memory unit, wherein a data set transfered between said shared memory unit and at least one member selected from the group consisiting of said first system and said second system is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated. Another embodiment of the invention is based on an apparatus comprising: a switch; a first system coupled to said switch; and a second system node coupled to said switch, wherein a data set transfered from said first system to said second system through said switch is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated.

These, and other, aspects of the present invention will be better appreciated and understood when considered in conjunction with the following description and the accompanying drawings. It should be understood, however, that the following description, while indicating preferred embodiments of the present invention and numerous specific details thereof, is given by way of

illustration and not of limitation. Many changes and modifications may be made within the scope of the present invention without departing from the spirit thereof, and the invention includes all such modifications.

## BRIEF DESCRIPTION OF THE DRAWINGS

5        A clear conception of the advantages and features constituting the present invention, and of the components and operation of model systems provided with the present invention, will become more readily apparent by referring to the exemplary, and therefore nonlimiting, embodiments illustrated

10      in the drawings accompanying and forming a part of this specification, wherein like reference numerals designate the same elements. It should be noted that the features illustrated in the drawings are not necessarily drawn to scale.

FIG. 1 illustrates a block schematic diagram of a network, representing an embodiment of the invention.

15      FIG. 2 illustrates a schematic diagram of a system architecture including a network switch, representing an embodiment of the invention.

FIG. 3 illustrates a block schematic diagram of a system architecture including a dedicated shared memory node device, representing an embodiment of the invention.

20      FIG. 4 illustrates a block schematic diagram of an interconnect fabric, representing an embodiment of the invention.

## DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention and the various features and advantageous details

25      thereof are explained more fully with reference to the nonlimiting embodiments that are illustrated in the accompanying drawings and detailed in the following description. Descriptions of well known components and processing techniques are omitted so as not to unnecessarily obscure the present invention in detail.

The teachings of U.S. Ser. No. 09/273,430 include a system which is a

30      single entity; one large supercomputer. The invention is also applicable to a cluster of workstations, or even a network.

The invention is applicable to systems of the type of Pfister or the type of U.S. Ser. No. 09/273,430 in which each processing node has its own copy of

an operating system. The invention is also applicable to other types of multiple processing node systems; even an interconnect fabric such as, for example, Infiniband.

The invention can be combined with a tight cluster as described in U.S. Ser. No. 09/273,430. A tight cluster is defined as a cluster of workstations or an arrangement within a single, multiple-processor machine in which the processors are connected by a high-speed, low-latency interconnection, and in which some but not all memory is shared among the processors. Within the scope of a given processor, accesses to a first set of ranges of memory addresses will be to local, private memory but accesses to a second set of memory address ranges will be to shared memory. The significant advantage to a tight cluster in comparison to a message-passing cluster is that, assuming the environment has been appropriately established, the exchange of information involves a single STORE instruction by the sending processor and a subsequent single LOAD instruction by the receiving processor.

The establishment of the environment, taught by U.S. Ser. No. 09/273,430 and more fully by companion disclosures (U.S. Provisional Application Ser. No. 60/220,794, filed July 26, 2000; U.S. Provisional Application Ser. No. 60/220,748, filed July 26, 2000; WSGR 15245-711; WSGR 15245-712; WSGR 15245-713; WSGR 15245-715; WSGR 15245-716; WSGR 15245-717; WSGR 15245-718; WSGR 15245-719; WSGR 15245-720, the entire contents of all which are hereby expressly incorporated herein by reference for all purposes) can be performed in such a way as to require relatively little system overhead, and to be done once for many, many information exchanges. Therefore, a comparison of 10,000 instructions for message-passing to a pair of instructions for tight-clustering, is valid.

The invention can include systems software to implement a low latency, high bandwidth multi-computer using existing readily commercially available commodity computer hardware and network devices. The invention can include a method to implement system software support for harnessing multiple, independent compute nodes using existing readily commercially available systems and network equipment or an interconnect fabric.

In general, the invention can include the use of a network driver shim

between a network driver layer, and a protocol software layer. The shim passes packets from the protocol software layer through to the network driver layer. Similarly, packets received from the network driver layer side are passed up to the protocol software layer.

5          A particular packet type identification can be used to decide how to handle received packets. As an example, in the case of the TCP/IP protocol, the Ethernet type identifier is 0x80-0x00, and is used by the shim to decide to pass the packet up to the protocol software layer for proper handling. In the case of low-latency packets taught by this invention, the shim can decide how best to

10        handle the packet. The invention can include transformation of a data set. For some cases, the shim can also implement a lightweight protocol in order to recover from errors encountered on the network media (such as CRC errors, hung network controllers, dropped packets, buffer errors, etc.). The advantages of the invention include improved cost/performance over existing proprietary

15        solutions.

          The shim can expose an API (application program interface) for transport middle-ware to use in order to transmit packets, obtain information on local and remote multi-computer nodes, to setup packet receive sinks, and to control the lightweight protocol. Fault tolerance can be achieved by ganging

20        multiple network interface cards in a single system, and either duplicating traffic over multiple network interface cards in a single system, or failing over when a failed NIC or system is detected. Fast recovery methods can be implemented by using network cards which give media sense interrupt indications, or by using relatively frequent "heartbeat" packets across the media.

25        Referring to FIG. 1, the invention can be implemented in the context of a network. A first network device/driver 110 is coupled to a network 100. A first shim 120 is coupled to the first network device/driver 110. A first protocol layer 130 is coupled to the first shim 120. The first shim 120 and the first protocol layer 130 can both interface with a first transport application program

30        interface (API) 135.

          Still referring to FIG. 1, a second network device/driver 140 is coupled to the network 100. A second shim 150 is coupled to the second network device/driver 140. A second protocol layer 160 is coupled to the second shim

150. The second shim 150 and the second protocol layer 160 can both interface with a second transport API 165.

The shims 120, 150 permit handling of data (e.g., routing and/or transformation) based on the type of data and/or the type of application associated with the transport APIs 135 and 165. The transport APIs may be for the same, or different, applications.

Referring to FIGS. 2-3, different types of system interconnects may be used. One example is the use of a true peer-to-peer interconnect through a network interconnect fabric (such as network switch). FIG. 2 depicts this arrangement. A system 0, a system 1, a system 2 and a system n-1 are all coupled to a network swich 200. System-to-system communication is accomplished through network communication provided by the network interface cards, media and network communications devices in the network.

Another system architecture that makes use of this capability is comprised of multiple compute nodes interconnected through a dedicated shared memory device. This model utilizes a "load-store" approach to remote memory access rather than message passing. This method reduces the cost associated with using a network communications switching fabric, and provides each system with a low latency, high bandwidth path to memory that is accessible by each compute node present in a particular configuration. An example of such a system structure is depicted in FIG. 3. In this embodiment, the system 0, the system 1, the system 2 and the system n-1 are all coupled to a dedicated shared memory node device 300. The dedicated shared memory node device may be RAM and/or a disk.

The system architecture of the invention may be used to implement any or all of the following subsystems:

1.    Network access through shared memory.

2.    A shared memory disk, where each system's backing store may be cached, and available in the dedicated shared memory node device.

3.    Locking primitives for controlled access to shared regions of memory.

Having a portion of shared memory common to each system allows each of the individual systems to have access to their own memory without the

normal overhead of cache coherency mechanisms usually used for tightly-coupled, shared memory multiprocessor systems.

Referring to FIG. 4, the invention can be implemented in the context of an interconnect fabric. A first interconnect fabric interface 410 is coupled to an interconnect fabric 400. A first shim 420 is coupled to the first interconnect fabric interface 410. A first protocol layer 430 is coupled to the first shim 420. The first shim 420 and the first protocol layer 430 can both interface with a first transport application program interface (API) 435.

Still referring to FIG. 4, a second interconnect fabric interface 440 is coupled to the network 400. A second shim 450 is coupled to the second interconnect fabric interface 440. A second protocol layer 460 is coupled to the second shim 450. The second shim 450 and the second protocol layer 460 can both interface with a second transport API 465.

Again, the shims 420, 450 permit handling of data (e.g., routing and/or transformation) based on the type of data and/or the type of application associated with the transport APIs 435 and 465. Again, the transport APIs may be for the same, or different, applications.

The context of the invention can include multi-computing. The context of the invention can include fault tolerance. The context of the invention can include shared-system network access. The context of the invention can include a shared network. The invention can include a network driver shim. The context of the invention can include an interconnect fabric, such as, for example, Infiniband.

The invention is an improvement over current clustering implementations in that traffic is intercepted and acted upon at the network device driver layer, and sent at the network device driver layer, and the invention also allows existing communication protocols to still use the same media. This provides a cost/performance benefit to the end customer.

This invention can be primarily systems software. Hardware accelerations can be applied by selecting network interface cards, which provide programmable packet type identification, and automatic media sense detection indications.

The invention can be implemented in the context of an ethernet network.

The ethernet can be connected to each of a plurality of PC machines by a NIC card (network interface card) inside each PC. A NIC has its own required appllication interface (API). NIC's are intended to pass messages between PC's. · These messages tend to be somewhat long and somewhat infrequent, so are not

5      well suited for shared memory, which is why the preferred design does not use NIC's . Additionally, they tend to be very simple, which means that more processing is required in the software.

The invention can include a device driver which presents an API to the OS and also does all of the processing NICs require. The invention can then

10    also present the data to the NIC using its require API (the "transport API"). The invention permit a shared-memory machine to be run over a standard network, albeit slower than the machine disclosed in U.S. Ser. No. 09/273,430. Certain applications may not have many LOADS and STORES to shared memory, in which case they will run about as well over a standard set of PC's with industry

15    standard network interconnections as they will on the hardware disclosed in U.S. Ser. No. 09/273,430.

The invention can also be implemented in the context of an interconnect fabric where a separate processor with some of its own memory is provided on a NIC. An example of an appropriate interconnect fabric is Infiniband. In this

20    way, a much simpler method can be defined by which a main processor, when it needs to send or receive some data, just presents a special, short descriptor to the processor on the NIC and lets this NIC processor actually GET or PUT the data.

While not being limited to any particular performance indicator or

25    diagnostic identifier, preferred embodiments of the invention can be identified one at a time by testing for the substantially highest performance. The test for the substantially highest performance can be carried out without undue experimentation by the use of a simple and conventional benchmark (speed) experiment.

30    The term substantially, as used herein, is defined as at least approaching a given state (e.g., preferably within 10% of, more preferably within 1% of, and most preferably within 0.1% of). The term coupled, as used herein, is defined as connected, although not necessarily directly, and not necessarily mechanically.

The term means, as used herein, is defined as hardware, firmware and/or software for achieving a result. The term program or phrase computer program, as used herein, is defined as a sequence of instructions designed for execution on a computer system. A program may include a subroutine, a function, a

5      procedure, an object method, an object implementation, an executable application, an applet, a servlet, a source code, an object code, and/or other sequence of instructions designed for execution on a computer system.

EXAMPLE

A specific embodiment of the present invention will now be further

10     described by the following, nonlimiting example which will serve to illustrate in some detail various features of significance. The example is intended merely to facilitate an understanding of ways in which the present invention may be practiced and to further enable those of skill in the art to practice the present invention. Accordingly, the examples should not be construed as limiting the

15     scope of the present invention.

The printed source code attached to this invention disclosure is an example of how this invention would be implemented on Windows NT 4.0 and an Intel or Intel compatible processor based personal computer, using the NDIS intermediate driver model. This example is intended to be exemplary, and does

20     not preclude an implementation on a different system, operating system, or type of network. This example also does not exclude hardware accelerations for network controllers to enhance the capability of that controller for this application. A description of the attached software modules follows (this description is in the order that the files are presented in the appendix):

25     1.      D:\nt4ddk\src\timesn\tnsdrvr\sources - A makefile description for creating       the binary image.

2.      D:\nt4ddk\src\timesn\tnsdrvr\tnsemul.rc - A file for describing the resource information to be embedded in the binary image.

3.      D:\nt4ddk\arc\timesn\tnsdrvr\tnsemul.def - A file for describing the

30     exported functions of the final binary image.

4.      D:\nt4ddk\src\times\tnsdrvr\tnsif.h - Describes the constants and structures needed for an application to interface directly with the loaded, executing, binary image.

5.      D:\nt4ddk\src\timesn\tnsdrvr\tnsdef.h - Times N Systems Specific macros and constants.

6.      D:\nt4ddk\src\timesn\tnsdrvr\tnsdebug.h - Header file for describing function prototypes. Constants, structures, and macros needed for using debug services.

7.      D:\nt4ddk\src\timesn\tnsdrvr\tnsapi.h - Header file for describing the exported Times N Systems services for emulating a high-speed interconnect.

8.      D:\nt4ddk\src\timesn\tnsdrvr\tns.h - Structures, function prototypes, constants, and macros for the module in whole, including managing the object context, and interfacing to an existing, commodity network interface device.

9.      D:\nt4ddk\src\timesn\tnsdrvr\tnsdebug.c - Debug services

10.     D:\nt4ddk\src\timesn\tnsdrvr\tnsapi.c - Implementations for the Times N Systems application programming interfaces for an emulated high-speed interconnect.

11.     D:\nt4ddk\src\timesn\tnsdrvr\tnsemul.c - Main initialization file, Driver entry, relatively infrequently used functions

12.     D:\nt4ddk\src\timesn\tnsdrvr\recv.c - Receive packet processing, including high-speed interconnect transport processing

13.     D:\nt4ddk\src\timesn\tnsdrvr\send.c -Send packet processing

1.      D:\nt4ddk\src\timesn\tnsclien\tnsclien.h - Client driver header file

2.      D:\nt4ddk\src\timesn\tnsclien\tnsclient.c - Client driver implementation (an example of how interconnect transport services would be used).

An experimental system was prototyped using 100Mbit/sec full and half-duplex network equipment, and gave very good throughput numbers.

Practical Applications of the Invention

A practical application of the invention that has value within the technological arts is waveform transformation. Further, the invention is useful in conjunction with data input and transformation (such as are used for the purpose of speech recognition), or in conjunction with transforming the appearance of a display (such as are used for the purpose of video games), or the like. There are virtually innumerable uses for the invention, all of which need not be detailed here.

## Advantages of the Invention

A system, representing an embodiment of the invention, can be cost effective and advantageous for at least the following reasons. The invention improves the speed of parallel computing systems. The invention improves the

5    scalability of parallel computing systems. The invention improves the overall system throughput for a system multi-computer implementation.

All the disclosed embodiments of the invention described herein can be realized and practiced without undue experimentation. Although the best mode of carrying out the invention contemplated by the inventor is disclosed above,

10    practice of the invention is not limited thereto. Accordingly, it will be appreciated by those skilled in the art that the invention may be practiced otherwise than as specifically described herein.

For example, although the low latency, high bandwidth multi-computer system interconnect described herein can be a separate module, it will be

15    manifest that the low latency, high bandwidth multi-computer system interconnect may be integrated into the system with which it is associated. Furthermore, all the disclosed elements and features of each disclosed embodiment can be combined with, or substituted for, the disclosed elements and features of every other disclosed embodiment except where such elements

20    or features are mutually exclusive.

It will be manifest that various additions, modifications and rearrangements of the features of the invention may be made without deviating from the spirit and scope of the underlying inventive concept. It is intended that the scope of the invention as defined by the appended claims and their

25    equivalents cover all such additions, modifications, and rearrangements.

The appended claims are not to be interpreted as including means-plus-function limitations, unless such a limitation is explicitly recited in a given claim using the phrase "means for." Expedient embodiments of the invention are differentiated by the appended subclaims.

# Appendix

```
 1 !IF 0
 2 Copyright (c) 1989-1993 Microsoft Corporation
 3
 4 Module Name:
 5     sources.
 6
 7 Abstract:
 8     This file specifies the target component being built and the list of
 9     sources files needed to build that component.  Also specifies optional
10     compiler switches and libraries that are unique for the component being
11     built.
12 !ENDIF
13
14 MAJORCOMP=ntos
15 MINORCOMP=ndis
16
17 TARGETNAME=tnsemul
18 TARGETTYPE=EXPORT_DRIVER
19 TARGETPATH=$(BASEDIR)\lib
20
21 TARGETLIBS=$(BASEDIR)\lib\*\$(DDKBUILDENV)\ndis.lib
22
23 INCLUDES=$(BASEDIR)\inc;$(BASEDIR)\src\network\inc;..\inc
24
25 C_DEFINES=$(C_DEFINES) -DNDIS_MINIPORT_DRIVER
26 C_DEFINES=$(C_DEFINES) -DNDIS40
27 C_DEFINES=$(C_DEFINES) -DNDIS40_MINIPORT
28 C_DEFINES=$(C_DEFINES) -DBINARY_COMPATIBLE=0
29
30 MSC_WARNING_LEVEL=/W3 /WX
31
32 SOURCES=tnsemul.c   \
33     recv.c       \
34     send.c       \
35     tnsapi.c     \
36     tnsdebug.c   \
37     tnsemul.rc
38
39
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.rc**

```
 1 #include <windows.h>
 2 #include <ntverp.h>
 3
 4 /*-------------------------------------------------*/
 5 /* the following lines are specific to this file */
 6 /*-------------------------------------------------*/
 7
 8 /* VER_FILETYPE, VER_FILESUBTYPE, VER_FILEDESCRIPTION_STR
 9  * and VER_INTERNALNAME_STR must be defined before including COMMON.VER
10  * The strings don't need a '\0', since common.ver has them.
11  */
12 #define VER_FILETYPE    VFT_DRV
13 #define VER_FILESUBTYPE VFT2_DRV_NETWORK
14 #define VER_FILEDESCRIPTION_STR    "Times N Systems Emulation Layer"
15 #define VER_INTERNALNAME_STR       "TNSEMUL.SYS"
16
17 #include "common.ver"
18
19 #include "evtmsg.rc"
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.def**                    **Page 1 of 1**

```
1 ; DEF File for TNSEMUL.SYS
2
3 NAME TNSEMUL.SYS
4
5 DESCRIPTION 'TNSEMUL.SYS'
6
7 EXPORTS
```

FII : D:\nt4DDK\src\timesn\tnsdrvr\tnsif.h                                    Page 1 of 1

```
  1  //***************************************************************************
  2  //
  3  // COPYRIGHT:
  4  //       This program is an unpublished work fully protected by the United
  5  //       States copyright laws and is considered a trade secret belonging to
  6  //       Times N Systems, Inc.  To the extent that this work may be
  7  //       considered "published", the following notice applies "(c) 1999  Times N
  8  //       Systems, Inc.".  Any unauthorized use, reproduction, distribution,
  9  //       display, modification, or disclosure of this program is strictly
 10  //       prohibited.
 11  //
 12  //***************************************************************************
 13  //
 14  //***************************************************************************
 15  // Module:
 16  //   tnsif.h - Times N Protocol Interface constants and structures
 17  //
 18  // DESCRIPTION:
 19  //
 20  // ENDTEXT:
 21  //
 22  //
 23  // Exports:
 24  //     See Module functions generated by script processing.
 25  //
 26  // AUTHOR:
 27  //       Vince Bridgers
 28  //       vince@timesn.com
 29  //
 30  //
 31  //***************************************************************************
 32  #ifndef _TNSIF_H_
 33  #define _TNSIF_H_
 34
 35  //
 36  // Debug levels
 37  //
 38  #define DEBUG_INFO        0
 39  #define DEBUG_MESSAGE     1
 40  #define DEBUG_WARNING     2
 41  #define DEBUG_VERBOSE     3
 42  #define DEBUG_ERROR       4
 43
 44  //
 45  // Debug mask bit definitions.  These are implemented as a bit mask
 46  // and are used to selectively enable/disable certain classes of debug
 47  // messages.
 48  //
 49  #define DEBUG_MASKEN_ERROR        0x01
 50  #define DEBUG_MASKEN_RECV         0x02
 51  #define DEBUG_MASKEN_SEND         0x04
 52  #define DEBUG_MASKEN_INIT         0x08
 53  #define DEBUG_MASKEN_PACKETDUMP   0x10
 54  #define DEBUG_MASKEN_ENTRYEXIT    0x20
 55
 56
 57  #define FILE_DEVICE_TNS  0x00008301
 58  #define TNS_IOCTL_BASE   0x830
 59  #define IOCTL_TNS_SETDEBUGINFO   CTL_CODE(FILE_DEVICE_TNS,      \
 60                                            TNS_IOCTL_BASE+0,      \
 61                                            METHOD_BUFFERED,       \
 62                                            (FILE_READ_ACCESS | FILE_WRITE_ACCESS))
 63
 64  typedef struct _TNS_IOCTLPACKET {
 65      ULONG    DebugLevel;
 66      ULONG    DebugMask;
 67      ULONG    DebugBreakFlag;
 68  } TNS_IOCTLPACKET, *pTNS_IOCTLPACKET;
 69
 70
 71  #endif
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdefs.h**          **Page 1 of 2**

```
 1
 2  //
 3  // COPYRIGHT:
 4  //    This program is an unpublished work fully protected by the United
 5  //    States copyright laws and is considered a trade secret belonging to
 6  //    Times-N Systems, Inc. To the extent that this work may be
 7  //    considered published the following notice applies: (c) 1999 Times N
 8  //    Systems, Inc. Any unauthorized use, reproduction, distribution,
 9  //    display, modification, or disclosure of this program is strictly
10  //    prohibited.
11  //
12  //
13  //
14  //
15  // Module:
16  //
17  // Description:
18  //
19  // Environment:
20  //
21  //
22  // Exports:
23  //
24  // Author:
25  //    Vince Bridgers
26  //    vinceb@timesn.com
27  //
28  //
29
30  typedef LONG          TNS_STATUS;
31  typedef TNS_STATUS *PTNS_STATUS;
32
33  typedef LONG          LOCKID;
34  typedef LOCKID       *PLOCKID;
35
36  typedef LONG          LOCKSTATUS;
37  typedef LOCKSTATUS   *PLOCKSTATUS;
38
39  typedef LONG          TNSKEY;
40  typedef TNSKEY       *PTNSKEY;
41
42  typedef LONG          TNSCPUID;
43  typedef TNSCPUID     *PTNSCPUID;
44
45  typedef LONG          TNSNOTIFYSTATUS;
46  typedef TNSNOTIFYSTATUS *PTNSNOTIFYSTATUS;
47
48
49  typedef LONG          TNSCOUNTER;
50  typedef TNSCOUNTER   *PTNSCOUNTER;
51
52  typedef LONG          TNSQUEUE;
53  typedef TNSQUEUE   *PTNSQUEUE;
54
55  typedef LONG             TNSQUEUEINFO;
56  typedef TNSQUEUEINFO    *PTNSQUEUEINFO;
57
58  typedef LONG    TNSMEMSIZE;
59
60  typedef LONG    TNSMEMFLAGS;
61
62
63  #define NTSTATUS_CUSTOMER_CODE  0x20000000
64
65  #define TNS_STATUS_CODE(Severity, StatusCode) (\
66      (NTSTATUS_CUSTOMER_CODE | (Severity << 30) | StatusCode))
67
68
69
70  //
71  //
72  //
73
74  typedef enum {
75      TNS_SUCCESS=0,
76      TNS_NOT_IMPLEMENTED,
77  };
78
79  #define TNS_STATUS_SUCCESS TNS_STATUS_CODE(STATUS_SEVERITY_SUCCESS,    TNS_SUCCESS)
80
81  #define TNS_STATUS_NOT_IMPLEMENTED TNS_STATUS_CODE(STATUS_SEVERITY_ERROR,    TNS_NOT_IMPLEMENTED)
82
```

83
84

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.h**                    **Page 1 of 2**

```
 1
 2  //
 3  // COPYRIGHT
 4  //      This program is an unpublished work fully protected by the United
 5  //      States copyright laws and is considered a trade secret belonging to
 6  //      Timesn Systems, Inc. To the extent that this work may be
 7  //      considered "published" the following notice applies: 1999 Times N
 8  //      Systems, Inc. Any unauthorized use, reproduction, distribution,
 9  //      display, modification, or disclosure of this program is strictly
10  //      prohibited.
11  //
12
13  //
14
15  // Module:
16  //      tnsdebug.h - timesN protocol debug support functions and the like
17  //
18  // Description:
19  //
20  // Environment:
21  //      timesN
22  //
23  // Exports:
24  //      See Module functions generated by script processing.
25  //
26  // Author:
27  //      Jinbo Bridgers
28  //      jinbo@timesn.com
29  //
30  //
31
32  #ifndef _TNSDEBUG_H_
33  #define _TNSDEBUG_H_
34
35  //
36  // TNS function to make a beep noise, for debugging occasionally
37  //
38  void
39  TNSMakeBeep(void);
40
41
42  #include "tnsif.h"
43
44
45  // Example of how to use for compile timesn reminders.
46  //
47  // #pragma message(REMIND("put stuff here"))
48  //
49
50  #define DEBUG_QUOTE(x) #x
51  #define DEBUG_QQUOTE(y) DEBUG_QUOTE(y)
52  #define REMIND(sz) __FILE__ "("DEBUG_QQUOTE(__LINE__)"):"sz
53
54  #ifdef DBG
55
56  char *GetNDISOidString(NDIS_OID NdisOID, PULONG pFoundFlag);
57  char *GetNDISStatusString(NDIS_STATUS Status, PULONG pFoundFlag);
58  char *GetNDISEventString(NDIS_ERROR_CODE ErrorCode, PULONG pFoundFlag);
59
60      VOID
61      NdisDumpPacket(
62          PNDIS_PACKET Packet);
63
64      #define STATIC
65
66      VOID
67      DebugPrint(
68          ULONG DebugPrintLevel,
69          PCSZ DebugMessage,
70          ...
71      );
72
73      VOID
74      MaskDebugPrint(
75          ULONG DebugPrintLevel,
76          ULONG DebugPrintMask,
77          PCSZ DebugMessage,
78          ...
79      );
80
81      extern ULONG _gDebugPrintLevel;
82      extern ULONG _gDebugPrintMask;
```

```
83      extern ULONG _gDebugBreakFlag;
84
85      #define DEBUG_MODULE "DEBUG: "
86
87      #define DINFO(x, y) \
88      DebugPrint(x, "%s", DEBUG_MODULE); \
89      DebugPrint(x, "File => %s: ", __FILE__); \
90      DebugPrint(x, "Line => %d: ", __LINE__); \
91      DebugPrint y;
92
93      #define D(x)  DebugPrint x;
94
95      #define DM(x) MaskDebugPrint x;
96
97      #define DUMP_PACKET(x) NdisDumpPacket(x)
98
99      #define INT3 { _asm int 3 }
100
101     #define BreakPoint() \
102     { DbgPrint("Debug Break in file => %s, at line %d\n", __FILE__, __LINE__); \
103       if (_gDebugBreakFlag) { _asm int 3 } ; }
104
105     #define MyAssert(c) if (!(c)) {\
106     { DbgPrint("Assertion failure: Debug Break in file => %s, at line %d\n", __FILE__, __LINE__); \
107       if (_gDebugBreakFlag) { _asm int 3 } ; } }
108
109 #else //_DBG
110
111     #define STATIC static
112     #define DINFO(x,y)
113     #define D(x)
114     #define DM(x)
115     #define BreakPoint()
116     #define INT3
117     #define MyAssert(c)
118     #define DUMP_PACKET(x)
119
120 #endif //_DBG
121 #endif //_TNSDEBUG_H
122
123
124
```

22

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h                    Page 1 of 11

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31  #define DECLSPEC_EXPORT _declspec(dllexport)
32
33
34
35  ULONG
36  DECLSPEC_EXPORT
37  __TNS_READ_REGISTER_ULONG(
38      IN  PVOID   DeviceHandle,
39      IN  PULONG  Register);
40
41
42
43
44
45
46
47
48
49
50
51
52  VOID
53  DECLSPEC_EXPORT
54  __TNS_WRITE_REGISTER_ULONG(
55      IN  PVOID   DeviceHandle,
56      IN  PULONG  Register,
57      IN  ULONG   RegisterData);
58
59
60
61
62
63
64
65
66
67
68
69
70  USHORT
71  DECLSPEC_EXPORT
72  __TNS_READ_REGISTER_USHORT(
73      IN  PVOID   DeviceHandle,
74      IN  PUSHORT Register);
75
76
77
78
79
80
81
82
```

```
 83  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
 84
 85  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
 86  ▓▓▓▓
 87  VOID
 88  DECLSPEC_EXPORT
 89  __TNS_WRITE_REGISTER_USHORT(
 90      IN  PVOID   DeviceHandle,
 91      IN  PUSHORT Register,
 92      IN  USHORT  RegisterData);
 93  ▓▓
 94  ▓▓▓Description▓
 95  ▓▓
 96  ▓▓Environment▓
 97  ▓▓
 98  ▓▓Return Value▓
 99  ▓▓▓▓▓
100  ▓▓▓
101  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
102
103  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
104  ▓▓▓
105  UCHAR
106  DECLSPEC_EXPORT
107  __TNS_READ_REGISTER_UCHAR(
108      IN  PVOID   DeviceHandle,
109      IN  PUCHAR  Register);
110  ▓▓
111  ▓▓Description▓
112  ▓▓
113  ▓▓Environment▓
114  ▓▓
115  ▓▓Return Value▓
116  ▓▓▓▓▓
117  ▓▓▓
118  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
119
120  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
121  ▓▓▓
122  VOID
123  DECLSPEC_EXPORT
124  __TNS_WRITE_REGISTER_UCHAR(
125      IN  PVOID   DeviceHandle,
126      IN  PUCHAR  Register,
127      IN  UCHAR   RegisterData);
128  ▓▓
129  ▓▓Description▓
130  ▓▓
131  ▓▓Environment▓
132  ▓▓
133  ▓▓Return Value▓
134  ▓▓▓▓▓
135  ▓▓▓
136  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
137
138
139
140  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
141  ▓▓▓
142  VOID
143  DECLSPEC_EXPORT
144  __TNS_READ_REGISTER_BUFFER_ULONG(
145      IN  PVOID   DeviceHandle,
146      IN  PULONG  Register,
147      IN  PULONG  pulBuffer,
148      IN  ULONG   Count);
149  ▓▓
150  ▓▓Description▓
151  ▓▓
152  ▓▓Environment▓
153  ▓▓
154  ▓▓Return Value▓
155  ▓▓▓▓▓
156  ▓▓▓
157  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
158
159  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
160  ▓▓▓
161  VOID
162  DECLSPEC_EXPORT
163  __TNS_WRITE_REGISTER_BUFFER_ULONG(
164      IN  PVOID   DeviceHandle,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapl.h**

```
165     IN  PULONG  Register,
166     IN  PULONG  pulBuffer,
167     IN  ULONG   Count);
168
169
170
171
172
173
174
175
176
177
178
179
180 VOID
181 DECLSPEC_EXPORT
182 _TNS_READ_REGISTER_BUFFER_USHORT(
183     IN  PVOID   DeviceHandle,
184     IN  PUSHORT Register,
185     IN  PUSHORT pusBuffer,
186     IN  ULONG   Count);
187
188
189
190
191
192
193
194
195
196
197
198
199 VOID
200 DECLSPEC_EXPORT
201 _TNS_WRITE_REGISTER_BUFFER_USHORT(
202     IN  PVOID   DeviceHandle,
203     IN  PUSHORT Register,
204     IN  PUSHORT pusBuffer,
205     IN  ULONG   Count);
206
207
208
209
210
211
212
213
214
215
216
217
218
219 VOID
220 DECLSPEC_EXPORT
221 _TNS_READ_REGISTER_BUFFER_UCHAR(
222     IN  PVOID   DeviceHandle,
223     IN  PUCHAR  Register,
224     IN  PUCHAR  pucBuffer,
225     IN  ULONG   Count);
226
227
228
229
230
231
232
233
234
235
236
237
238 VOID
239 DECLSPEC_EXPORT
240 _TNS_WRITE_REGISTER_BUFFER_UCHAR(
241     IN  PVOID   DeviceHandle,
242     IN  PUCHAR  Register,
243     IN  PUCHAR  pucBuffer,
244     IN  ULONG   Count);
245
246
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h                    Page 4 of 11

```
247
248
249
250
251
252
253
254
255
256
257 TNS_STATUS
258 DECLSPEC_EXPORT
259 __TNSAcquireLockP(
260     IN  PVOID   DeviceHandle,
261     IN  PLOCKID pLockID);
262
263
264
265
266
267
268
269
270
271
272
273
274 TNS_STATUS
275 DECLSPEC_EXPORT
276 __TNSReleaseLockP(
277     IN  PVOID   DeviceHandle,
278     IN  PLOCKID pLockID);
279
280
281
282
283
284
285
286
287
288
289
290
291 TNS_STATUS
292 DECLSPEC_EXPORT
293 __TNSQueryLockP(
294     IN  PVOID        DeviceHandle,
295     OUT PLOCKSTATUS pLockStatus);
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310 TNS_STATUS
311 DECLSPEC_EXPORT
312 __TNSAllocateLockP(
313     IN  PVOID   DeviceHandle,
314     IN  TNSKEY  Key,
315     OUT PLOCKID *pLockID);
316
317
318
319
320
321
322
323
324
325
326
327
328 TNS_STATUS
```

```
329 DECLSPEC_EXPORT
330 __TNSFreeLockP(
331     IN  PVOID    DeviceHandle,
332     IN  TNSKEY   Key,
333     IN  PLOCKID  pLockID);
334
335
336
337
338
339
340
341
342
343
344
345
346 TNS_STATUS
347 DECLSPEC_EXPORT
348 __TNSNotifyCPU(
349     IN  PVOID     DeviceHandle,
350     IN  TNSCPUID  CpuID,
351     IN  PVOID     pMessageBuffer,
352     IN  ULONG     MessageLength);
353
354
355
356
357
358
359
360
361
362
363
364
365 TNS_STATUS
366 DECLSPEC_EXPORT
367 __TNSNotifyCPUSync(
368     IN  PVOID     DeviceHandle,
369     IN  TNSCPUID  CpuID,
370     IN  PVOID     pMessageBuffer,
371     IN  ULONG     MessageLength,
372     IN  PVOID     pCallback,
373     IN  PVOID     pContext);
374
375
376
377
378
379
380
381
382
383
384
385
386
387 TNS_STATUS
388 DECLSPEC_EXPORT
389 __TNSQueryNotifyStatus(
390     IN     PVOID             DeviceHandle,
391     IN     TNSCPUID          CpuID,
392     IN OUT PTNSNOTIFYSTATUS  pCpuNotifyInfo);
393
394
395
396
397
398
399
400
401
402
403
404
405
406 TNS_STATUS
407 DECLSPEC_EXPORT
408 __TNSRegisterNotifyCallback(
409     IN  PVOID    DeviceHandle,
410     IN  PVOID    pCallBack,
```

FII : D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h                    Pag  6 of 11

```
411      IN   PVOID       SysParm1,
412      IN   PVOID       SysParm2,
413      IN   PVOID       SysParm3);
414  //
415  // Description:
416  //
417  // Environment:
418  //
419  // Return Value:
420  //
421  //
422  //
423
424
425  //
426  //
427  TNS_STATUS
428  DECLSPEC_EXPORT
429  _TNSRegisterNotificationCallback(
430      IN   PVOID       DeviceHandle,
431      IN   PVOID       pCallBack,
432      IN   PVOID       SysParm1,
433      IN   PVOID       SysParm2,
434      IN   PVOID       SysParm3);
435  //
436  // Description:
437  //
438  // Environment:
439  //
440  // Return Value:
441  //
442  //
443  //
444
445
446  //
447  //
448  TNS_STATUS
449  DECLSPEC_EXPORT
450  _TNSDeRegisterNotificationCallback(
451      IN   PVOID       DeviceHandle,
452      IN   PVOID       pCallBack);
453  //
454  // Description:
455  //
456  // Environment:
457  //
458  // Return Value:
459  //
460  //
461  //
462
463
464  //
465  //
466  TNSCPUID
467  DECLSPEC_EXPORT
468  _TNSWhoAmI(
469      IN   PVOID       DeviceHandle);
470  //
471  // Description:
472  //
473  // Environment:
474  //
475  // Return Value:
476  //
477  //
478  //
479
480  //
481  //
482  TNSCOUNTER
483  DECLSPEC_EXPORT
484  _TNSReadOrdinalCounter(
485      IN   PVOID       DeviceHandle);
486  //
487  // Description:
488  //
489  // Environment:
490  //
491  // Return Value:
492  //
```

File: D:\nt4DDK\src\tlmesn\tnsdrvr\tnsapl.h                    Page 7 of 1·

```
493
494
495
496
497
498
499  TNS_STATUS
500  DECLSPEC_EXPORT
501  __TNSAllocateSharedMemory(
502      IN      PVOID       DeviceHandle,
503      IN      TNSKEY      Key,
504      IN      TNSMEMFLAGS Flags,
505      IN      TNSMEMSIZE  Size,
506      IN OUT  PVOID       *ppBuffer);
507
508
509
510
511
512
513
514
515
516
517
518
519
520  TNS_STATUS
521  DECLSPEC_EXPORT
522  __TNSFreeSharedMemory(
523      IN  PVOID       DeviceHandle,
524      IN  TNSKEY      Key,
525      IN  PVOID       Ptr,
526      IN  TNSMEMSIZE  Size);
527
528
529
530
531
532
533
534
535
536
537
538
539  TNS_STATUS
540  DECLSPEC_EXPORT
541  __TNSReadSharedMemory(
542      IN  PVOID       DeviceHandle,
543      IN  PVOID       pSharedMemoryAddress,
544      IN  ULONG       Length,
545      IN  PVOID       pBuffer);
546
547
548
549
550
551
552
553
554
555
556
557
558
559  TNS_STATUS
560  DECLSPEC_EXPORT
561  __TNSWriteSharedMemory(
562      IN  PVOID       DeviceHandle,
563      IN  PVOID       pSharedMemoryAddress,
564      IN  ULONG       Length,
565      IN  PVOID       pBuffer);
566
567
568
569
570
571
572
573
574
```

```
575
576  ///////////////////////////////////////////////////////////////
577  //--
578  TNS_STATUS
579  DECLSPEC_EXPORT
580  __TNSDmaReadSharedMemory(
581     IN   PVOID      DeviceHandle,
582     IN   PVOID      pSharedMemoryAddress,
583     IN   ULONG      Length,
584     IN   PVOID      pBuffer,
585     IN   PVOID      pCallback,
586     IN   PVOID      DMAReadCompleteComtext1,
587     IN   PVOID      DMAReadCompleteComtext2);
588  //;
589  // Description:
590  //;
591  // Arguments:
592  //;
593  // Return value:
594  //;
595  //--
596  ///////////////////////////////////////////////////////////////
597
598  ///////////////////////////////////////////////////////////////
599  //--
600  TNS_STATUS
601  DECLSPEC_EXPORT
602  __TNSDmaWriteSharedMemory(
603     IN   PVOID      DeviceHandle,
604     IN   PVOID      pSharedMemoryAddress,
605     IN   ULONG      Length,
606     IN   PVOID      pBuffer,
607     IN   PVOID      pCallback,
608     IN   PVOID      DMAWriteCompleteComtext1,
609     IN   PVOID      DMAWriteCompleteComtext2);
610  //;
611  // Description:
612  //;
613  // Arguments:
614  //;
615  // Return value:
616  //;
617  //--
618  ///////////////////////////////////////////////////////////////
619
620  ///////////////////////////////////////////////////////////////
621  //--
622  TNS_STATUS
623  DECLSPEC_EXPORT
624  __TNSAllocateWorkQueue(
625     IN       PVOID       DeviceHandle,
626     IN       TNSKEY      Key,
627     IN       PULONG      pQueueLength,
628     IN OUT   PTNSQUEUE   *ppTNSQueue);
629  //;
630  // Description:
631  //;
632  // Arguments:
633  //;
634  // Return value:
635  //;
636  //--
637  ///////////////////////////////////////////////////////////////
638
639
640  ///////////////////////////////////////////////////////////////
641  //--
642  TNS_STATUS
643  DECLSPEC_EXPORT
644  __TNSFreeWorkQueue(
645     IN       PVOID       DeviceHandle,
646     IN       TNSKEY      Key,
647     IN       PTNSQUEUE   pTNSQueue);
648  //;
649  // Description:
650  //;
651  // Arguments:
652  //;
653  // Return value:
654  //;
655  //--
656  ///////////////////////////////////////////////////////////////
```

```
657
658  //================================================================
659  //==
660  TNS_STATUS
661  DECLSPEC_EXPORT
662  __TNSInterlockedEnqueueToDoP(
663      IN      PVOID       DeviceHandle,
664      IN      PTNSQUEUE   pTNSQueue,
665      IN      PVOID       pItem,
666      IN      ULONG       Length);
667  //
668  // Description:
669  //
670  // Environment:
671  //
672  // Return value:
673  //
674  //==
675  //================================================================
676
677
678  //================================================================
679  //==
680  TNS_STATUS
681  DECLSPEC_EXPORT
682  __TNSInterlockedDequeueToDoP(
683      IN      PVOID       DeviceHandle,
684      IN      PTNSQUEUE   pTNSQueue,
685      IN      PVOID       pItem,
686      IN      PULONG      pLength);
687  //
688  // Description:
689  //
690  // Environment:
691  //
692  // Return value:
693  //
694  //==
695  //================================================================
696
697  //================================================================
698  //==
699  TNS_STATUS
700  DECLSPEC_EXPORT
701  __TNSQueryQLengthP(
702      IN      PVOID       DeviceHandle,
703      IN      PTNSQUEUE   pTNSQueue,
704      IN      PULONG      pLength);
705  //
706  // Description:
707  //
708  // Environment:
709  //
710  // Return value:
711  //
712  //==
713  //================================================================
714
715
716  //================================================================
717  //==
718  TNS_STATUS
719  DECLSPEC_EXPORT
720  __TNSQueueHeadP(
721      IN      PVOID       DeviceHandle,
722      IN      PTNSQUEUE   pTNSQueue,
723      IN OUT  PTNSQUEUE   *ppTNSQueue);
724  //
725  // Description:
726  //
727  // Environment:
728  //
729  // Return value:
730  //
731  //==
732  //================================================================
733
734
735  //================================================================
736  //==
737  TNS_STATUS
738  DECLSPEC_EXPORT
```

```
739  __TNSQueueTailP(
740      IN       PVOID       DeviceHandle,
741      IN       PTNSQUEUE   pTNSQueue,
742      IN OUT   PTNSQUEUE   *ppTNSQueue);
743  //
744  //  Description:
745  //
746  //  Comments:
747  //
748  //  Return Value:
749  //
750  //
751  //
752
753
754  //
755  //
756  TNS_STATUS
757  DECLSPEC_EXPORT
758  __TNSQueuePayloadP(
759      IN       PVOID       DeviceHandle,
760      IN       PTNSQUEUE   pTNSQueue,
761      IN       PVOID       pItem,
762      IN       PULONG      pLength);
763  //
764  //  Description:
765  //
766  //  Comments:
767  //
768  //  Return Value:
769  //
770  //
771  //
772
773
774  //
775  //
776  TNS_STATUS
777  DECLSPEC_EXPORT
778  __TNSQueueNextP(
779      IN       PVOID       DeviceHandle,
780      IN       PTNSQUEUE   pTNSQueue,
781      IN OUT   PTNSQUEUE   *ppTNSQueue);
782  //
783  //  Description:
784  //
785  //  Comments:
786  //
787  //  Return Value:
788  //
789  //
790  //
791
792  //
793  //
794  TNS_STATUS
795  DECLSPEC_EXPORT
796  __TNSInterlockedInsertQueueItemP(
797      IN       PVOID       DeviceHandle,
798      IN       PTNSQUEUE   pTNSQueue,
799      IN       PTNSQUEUE   pTNSQueueInsert);
800  //
801  //  Description:
802  //
803  //  Comments:
804  //
805  //  Return Value:
806  //
807  //
808  //
809  .
810
811  //
812  //
813  TNS_STATUS
814  DECLSPEC_EXPORT
815  __TNSInterlockedDeleteQueueItemP(
816      IN       PVOID       DeviceHandle,
817      IN       PTNSQUEUE   pTNSQueue,
818      IN       PTNSQUEUE   pTNSQueueDelete);
819  //
820  //  Description:
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h**                    **Page 11 of 11**

```
821  //
822  // Environments
823  //
824  // Return Value:
825  //
826  //
827  //
828
829
830  //
831  TNS_STATUS
832  DECLSPEC_EXPORT
833  __TNSQueueItemInfoP(
834      IN      PVOID            DeviceHandle,
835      IN      PTNSQUEUE        pTNSQueue,
836      IN      PTNSQUEUEINFO    pTNSQueueInfo);
837  //
838  // Description:
839  //
840  // Environment:
841  //
842  // Return Value:
843  //
844  //
845  //
846
847
848  TNS_STATUS
849  DECLSPEC_EXPORT
850  __TNSGetFirstDeviceInstance(
851      PVOID    *ppDeviceInstance);
852
853  TNS_STATUS
854  DECLSPEC_EXPORT
855  __TNSGetNextDeviceInstance(
856      PVOID    pDeviceInstance,
857      PVOID    *ppDeviceInstance);
858
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h**                          **Page 1 of 11**

```
 1  //*******************************************************************
 2  //
 3  // COPYRIGHT:
 4  //     This program is an unpublished work fully protected by the United
 5  //     States copyright laws and is considered a trade secret belonging to
 6  //     Times N Systems, Inc. To the extent that this work may be
 7  //     considered "published," the following notice applies: (c) 1999, Times N
 8  //     Systems, Inc.  Any unauthorized use, reproduction, distribution,
 9  //     display, modification, or disclosure of this program is strictly
10  //     prohibited.
11  //
12  //*******************************************************************
13  //--
14  //*******************************************************************
15  // Module:
16  //     tns.h - Times N Protocol packet definition for emulated subsystem
17  //
18  // Description:
19  //
20  // Environment:
21  //
22  // Exports:
23  //     See Module functions generated by script processing.
24  //
25  // Author:
26  //     Vince Bridgers
27  //     vinceb@timesn.com
28  //
29  //*******************************************************************
30
31  #ifndef _TNS_H_
32  #define _TNS_H_
33  #include <ntddk.h>
34  #include <ndis.h>
35  #include <ntddndis.h>
36  #include <tdikrnl.h>
37  #include "tnsstats.h"
38
39  #define MIN_PACKET_POOL_SIZE    0xff
40  #define MAX_PACKET_POOL_SIZE    0xffff
41
42  //
43  // Shutdown mask values
44  //
45
46  #define SHUTDOWN_DEALLOC_PACKET_POOL     0x00000001
47  #define SHUTDOWN_DEALLOC_LOOKAHEAD_POOL  0x00000002
48  #define SHUTDOWN_DEALLOC_RESIDUAL_POOL   0x00000004
49  #define SHUTDOWN_DEINIT_DEV_INSTANCE     0x00000008
50  #define SHUTDOWN_DELETE_PIPE             0x00000010
51  #define SHUTDOWN_TERMINATE_WRAPPER       0x00000040
52  #define SHUTDOWN_DEREGISTER_PROTOCOL     0x00000080
53  #define SHUTDOWN_DELETE_DEVICE           0x00000100
54  #define SHUTDOWN_DELETE_SYMLINK          0x00000200
55
56  #define READ_HIDDEN_CONFIG( _Field, ParamType ) \
57  {                                                              \
58      ConfigurationInfo->_Field =                                \
59          ReadSingleParameter(ConfigHandle,                      \
60                              Str ## _Field,                     \
61                              ConfigurationInfo->_Field,         \
62                              ParamType);                        \
63  }
64
65  #define DECLARE_STRING( _str_ ) STATIC WCHAR Str ## _str_[] = L#_str_
66
67  #define ETH_ADDRESS_LEN 6
68
69  //
70  // number of characters that are appended to the RegistryPath when constructing
71  // the miniport device name
72  //
73
74  #define MPNAME_EXTENSION_SIZE    ( 3 * sizeof(WCHAR))
75
76
77  #define MAX_COMPUTER_NAME_SIZE 16
78
79  typedef struct _SMNNodeTable {
80      int            LocationSet;
81      unsigned char  TNMacAddress[HARDWARE_ADDRESS_LENGTH];
82      unsigned long  TNNodeID;
```

```
 83      unsigned char TNComputerName[MAX_COMPUTER_NAME_SIZE];
 84  } SMNNodeTable, *pSMNNodeTable;
 85
 86  #define MAX_TEAM_NODES  128
 87
 88  //
 89  // per Adapter control block
 90  //
 91  typedef struct _ADAPTER {
 92      //
 93      // Required structure member for using DDK provided list management
 94      // functions
 95      //
 96      LIST_ENTRY Linkage;
 97
 98
 99      BOOLEAN TNSDriverInitialized;
100
101      //
102      // Size of this struct plus allocated strings
103      //
104      int AdapterStructSize;
105
106      //
107      // structure book keeping
108      //
109      // TNSDeviceName, MPDeviceName - Unicode device names for the intermediate
110      // and underlying
111      // MP device. The buffers for the strings are allocated as part of the adapter
112      // structure allocation and are located just after the structure. Buffer size
113      // is fixed at DEVNAME_SIZE
114      //
115      // ShutdownMask - mask of operations to perform during unbinding from lower MP
116      //
117
118      NDIS_STRING TNSDeviceName;
119      NDIS_STRING MPDeviceName;
120      ULONG ShutdownMask;
121      ULONG TNSMPState;
122      //
123      // miniport vars
124      //
125      // DevInstance - contains the number at the end of the device instance string
126      // such as "TmSampMP3". This is used by MPInitialize to determine which IM
127      // device is being initialized. Comparison via device names is not possible
128      // since the RTL unicode routines run at lowered IRQL and MPInitialize cannot
129      // guarantee that condition.
130      //
131      // CopyLookaheadData - TRUE if the BM can directly touch the lookahead data
132      //
133      // TNSNdisHandle - the handle that identifies the IM device to NDIS
134      //
135      // BlockingEvent - used to synchronize execution of functions that are
136      // awaiting completion
137      //
138      // FinalStatus - ndis status returned in completion routine
139      //
140      // PacketPoolHandle - handle to pool of NDIS_PACKETs used during Send and Packet
141      // recieve operations
142      //
143      // PacketList - a list of pre allocated packet structures
144      //
145      USHORT DevInstance;
146      BOOLEAN CopyLookaheadData;
147      NDIS_HANDLE TNSNdisHandle;
148      NDIS_EVENT BlockingEvent;
149      NDIS_STATUS FinalStatus;
150      NDIS_HANDLE PacketPoolHandle;
151
152      //
153      // Lookahead and residual buffer pool vars. Residual buffer size
154      // is same as TotalSize
155      //
156
157      ULONG LookaheadBufferSize;
158      NDIS_HANDLE LookaheadPoolHandle;
159
160      //
161      // underlying adapter info - handle, type etc.
162      //
163      // LowerMPHandle - the binding handle to the underlying MP
164      //
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h**           **Page 3 of 11**

```
165    //
166    //
167    //
168    //
169    //
170    //
171    //
172    //
173    //
174
175    NDIS_HANDLE LowerMPHandle;
176    UCHAR       LowerMPMacAddress[HARDWARE_ADDRESS_LENGTH];
177
178
179
180    NDIS_HANDLE BindContext;
181    NDIS_MEDIUM MediaType;
182    ULONG LinkSpeed;
183    ULONG TotalSize;
184    LIST_ENTRY ClientList;
185
186    //
187    //
188    //  Objects for managaging the client worker thread
189    //
190
191    ULONG       ListEntryItems;
192
193    HANDLE      ClientWorkerThreadHandle;
194    HANDLE      ServerWorkerThreadHandle;
195    //
196    //
197    //
198    KSPIN_LOCK  ListEntryPoolLock;
199    //
200    //
201    //
202    LIST_ENTRY  WorkerListEntryPool;
203
204    //
205    //
206    //
207    KSEMAPHORE  ClientWorkerRequestSemaphore;
208
209    //
210    //
211    //
212    KSEMAPHORE  ClientWorkerResponseSemaphore;
213
214    //
215    //
216    //
217    KSPIN_LOCK  ClientWorkerListSpinLock;
218    //
219    //
220    //
221    LIST_ENTRY  ClientWorkerListEntry;
222
223
224    //
225    //
226    //
227
228    //
229    //
230    //
231    KSEMAPHORE  ServerWorkerRequestSemaphore;
232    //
233    //
234    //
235    KSPIN_LOCK  ServerWorkerListSpinLock;
236    //
237    //
238    //
239    LIST_ENTRY  ServerWorkerListEntry;
240
241    UCHAR       SMNMacAddress[HARDWARE_ADDRESS_LENGTH];
242
243    //
244    //
245    //
246    NDIS_REQUEST Request;
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h                    **Page 4 of 11**

```
247    PULONG      BytesNeeded;
248    PULONG      BytesReadOrWritten;
249    BOOLEAN     LocalRequest;
250
251
252    PVOID       TNSSharedMemoryPtr;
253    ULONG       TNSSharedMemorySize;
254
255 #define VIRTUAL_MEMORY    1
256 #define NONPAGED_MEMORY   2
257
258    int         TNSMemoryType;
259
260    ULONG       TNSClientNodeID;
261            .
262    SMNNodeTable TeamNodeTable[MAX_TEAM_NODES];
263
264    STATISTICS  MyStats;
265    MPSTATS     mpStats;
266
267    KSPIN_LOCK  MyStatsLock;
268
269    unsigned char LocalComputerName[MAX_COMPUTER_NAME_SIZE];
270
271    unsigned char SMNMachineName[16];
272
273 } ADAPTER, *PADAPTER;
274
275 #define MAX_READWRITE_BUFFER_SIZE   1024
276
277
278 ▨
279 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
280 ▨
281
282 #define NdisRequestLocalSetInfo     NdisRequestGeneric1
283 #define NdisRequestLocalQueryInfo   NdisRequestGeneric2
284
285 ▨
286 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
287 ▨▨▨▨▨▨▨▨
288 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
289 ▨
290 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
291 ▨▨▨▨▨▨▨▨▨▨▨▨▨
292 ▨
293 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
294 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
295 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
296 ▨
297 typedef struct   _TNS_PACKET_CONTEXT {
298     PNDIS_PACKET OriginalPacket;
299     PNDIS_BUFFER LookaheadBuffer;
300     int          SMNEmulationPacket;
301 } TNS_PACKET_CONTEXT, *PTNS_PACKET_CONTEXT;
302
303 #define PACKET_CONTEXT_FROM_PACKET(_pkt ) ((PTNS_PACKET_CONTEXT)((_pkt)->ProtocolReserved))
304
305
306 #define MEDIA_INFO_SIZE     (sizeof( MEDIA_SPECIFIC_INFORMATION ) + sizeof( ULONG ))
307
308
309 ▨
310 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
311 ▨▨▨▨▨▨▨▨▨▨▨▨
312 ▨
313 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
314 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
315 ▨
316 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
317 ▨
318
319 typedef struct   _BUFFER_CONTEXT {
320     SINGLE_LIST_ENTRY SListEntry;
321     PNDIS_BUFFER NdisBuffer;
322 } BUFFER_CONTEXT, *PBUFFER_CONTEXT;
323
324 ▨
325 ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨
326 ▨
327
328 typedef struct _CONFIG_DATA {
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h**                            **Page 5 of 11**

```
329     ULONG PacketPoolSize;
330     ULONG DebugLevel;
331     ULONG DebugMask;
332     ULONG TNSSMNEmulationMode;
333 } CONFIG_DATA, *PCONFIG_DATA;
334
335 //
336 // values for error log entries
337 //
338
339 #define TNS_ERROR_MISSING_OID                       0x00010000
340 #define TNS_ERROR_BAD_REGISTRY_DATA                 0x00020000
341 #define TNS_ERROR_CANT_INITIALIZE_IMSAMP_DEVICE     0x00040000
342 #define TNS_ERROR_PACKET                            0x00060000
343 #define TNS_ERROR_PACKET_POOL                       0x00070000
344 #define TNS_ERROR_LOOKAHEAD_POOL                    0x00080000
345 #define TNS_ERROR_VM_LOOKAHEAD_BUFFER               0x00090000
346 #define TNS_ERROR_LOOKAHEAD_BUFFER                  0x000A0000
347 #define TNS_ERROR_RESIDUAL_POOL                     0x000B0000
348 #define TNS_ERROR_VM_RESIDUAL_BUFFER                0x000C0000
349 #define TNS_ERROR_RESIDUAL_BUFFER                   0x000D0000
350 #define TNS_ERROR_PROTOCOL_INIT                     0X000F0000
351
352 // bad registry data indicator
353
354 #define TNS_ERROR_INVALID_IMSAMP_MP_INSTANCE        0x00000004
355
356 //
357 // global vars (not based on a device instance)
358 //
359 extern ULONG TNSSharedMemoryNodeEmulation;
360
361 extern LIST_ENTRY AdapterList;
362 extern NDIS_SPIN_LOCK AdapterListLock;
363 extern NDIS_HANDLE ClientProtocolHandle;
364 extern NDIS_HANDLE MPWrapperHandle;
365 extern NDIS_HANDLE LMDriverHandle;
366 extern PDRIVER_OBJECT IMDriverObject;
367 extern PDEVICE_OBJECT IMDeviceObject;
368
369 extern CONFIG_DATA ConfigData;              // pointer to Registry Data
370
371 extern NDIS_STRING IMSymbolicName;
372 extern NDIS_STRING IMDriverName;
373 extern NDIS_STRING IMMPName;
374
375
376 VOID
377 MPSendPackets(
378     IN  NDIS_HANDLE             MiniportAdapterContext,
379     IN  PPNDIS_PACKET           PacketArray,
380     IN  UINT                    NumberOfPackets);
381
382 VOID
383 CLSendComplete(
384     IN  NDIS_HANDLE             ProtocolBindingContext,
385     IN  PNDIS_PACKET            Packet,
386     IN  NDIS_STATUS             Status);
387
388 VOID
389 PacketCompletion(
390     IN PADAPTER Adapter,
391     IN PNDIS_PACKET Packet,
392     IN NDIS_STATUS Status);
393
394 INT
395 CLReceivePacket(
396     IN  NDIS_HANDLE             ProtocolBindingContext,
397     IN  PNDIS_PACKET            Packet);
398
399 VOID
400 MPReturnPacket(
401     IN  NDIS_HANDLE             MiniportAdapterContext,
402     IN  PNDIS_PACKET            Packet);
403
404 NDIS_STATUS
405 CLReceiveIndication(
406     IN  NDIS_HANDLE             ProtocolBindingContext,
407     IN  NDIS_HANDLE             MacReceiveContext,
408     IN  PVOID                   HeaderBuffer,
409     IN  UINT                    HeaderBufferSize,
410     IN  PVOID                   LookAheadBuffer,
```

```
411      IN  UINT                    LookaheadBufferSize,
412      IN  UINT                    PacketSize);
413
414 VOID
415 CLReceiveComplete(
416      IN  NDIS_HANDLE             ProtocolBindingContext);
417
418 NDIS_STATUS
419 MPTransferData(
420      OUT PNDIS_PACKET            Packet,
421      OUT PUINT                   BytesTransferred,
422      IN  NDIS_HANDLE             MiniportAdapterContext,
423      IN  NDIS_HANDLE             MiniportReceiveContext,
424      IN  UINT                    ByteOffset,
425      IN  UINT                    BytesToTransfer);
426
427 VOID
428 CLTransferDataComplete(
429      IN  NDIS_HANDLE      ProtocolBindingContext,
430      IN  PNDIS_PACKET     pNdisPacket,
431      IN  NDIS_STATUS      Status,
432      IN  UINT             BytesTransferred);
433
434 VOID
435 BindToLowerMP(
436      OUT PNDIS_STATUS            Status,
437      IN  NDIS_HANDLE             BindContext,
438      IN  PNDIS_STRING            MPDeviceName,
439      IN  PVOID                   SystemSpecific1,
440      IN  PVOID                   SystemSpecific2);
441
442 VOID
443 LowerMPOpenAdapterComplete(
444      IN  NDIS_HANDLE ProtocolBindingContext,
445      IN  NDIS_STATUS Status,
446      IN  NDIS_STATUS OpenErrorStatus);
447
448 NDIS_STATUS
449 MPInitialize(
450      OUT PNDIS_STATUS            OpenErrorStatus,
451      OUT PUINT                   SelectedMediumIndex,
452      IN  PNDIS_MEDIUM            MediumArray,
453      IN  UINT                    MediumArraySize,
454      IN  NDIS_HANDLE             MiniportAdapterHandle,
455      IN  NDIS_HANDLE             WrapperConfigurationContext);
456
457 PADAPTER
458 FindAdapterByName(
459      PWCHAR AdapterName);
460
461 VOID
462 UnbindFromLowerMP(
463      OUT PNDIS_STATUS            Status,
464      IN  NDIS_HANDLE             ProtocolBindingContext,
465      IN  NDIS_HANDLE             UnbindContext);
466
467 VOID
468 DerefAdapter(
469      PADAPTER Adapter);
470
471 VOID
472 CleanupAdapter(
473      PADAPTER Adapter);
474
475 VOID
476 LowerMPCloseAdapterComplete(
477      IN  NDIS_HANDLE ProtocolBindingContext,
478      IN  NDIS_STATUS Status);
479
480 VOID
481 CLUnloadProtocol(
482      VOID);
483
484 VOID
485 MPHalt(
486      IN  NDIS_HANDLE             MiniportAdapterContext);
487
488 NDIS_STATUS
489 MPReset(
490      OUT PBOOLEAN                AddressingReset,
491      IN  NDIS_HANDLE             MiniportAdapterContext);
492
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h**                    **Page 7 of 11**

```
493
494 NDIS_STATUS
495 MPQueryInformation(
496     IN  NDIS_HANDLE         MiniportAdapterContext,
497     IN  NDIS_OID            Oid,
498     IN  PVOID               InformationBuffer,
499     IN  ULONG               InformationBufferLength,
500     OUT PULONG              BytesWritten,
501     OUT PULONG              BytesNeeded);
502
503 NDIS_STATUS
504 MPSetInformation(
505     IN  NDIS_HANDLE         MiniportAdapterContext,
506     IN  NDIS_OID            Oid,
507     IN  PVOID               InformationBuffer,
508     IN  ULONG               InformationBufferLength,
509·    OUT PULONG              BytesRead,
510     OUT PULONG              BytesNeeded);
511
512 VOID
513 CLRequestComplete(
514     IN  NDIS_HANDLE     ProtocolBindingContext,
515     IN  PNDIS_REQUEST   NdisRequestBuf,
516     IN  NDIS_STATUS     Status);
517
518 NDIS_STATUS
519 MakeLocalNdisRequest(
520     PADAPTER Adapter,
521     NDIS_OID Oid,
522     PVOID Buffer,
523     ULONG BufferSize);
524
525 NDIS_STATUS
526 MakeLocalNdisRequestSet(
527     PADAPTER Adapter,
528     NDIS_OID Oid,
529     PVOID Buffer,
530     ULONG BufferSize);
531
532                     .
533 NTSTATUS
534 WDMInitialize(
535     PDRIVER_OBJECT DriverObject,
536     PULONG InitShutdownMask);
537
538 VOID
539 WDMCleanup(
540     ULONG ShutdownMask);
541
542 NTSTATUS
543 ConfigureDriver (
544     IN PUNICODE_STRING RegistryPath,
545     IN PCONFIG_DATA ConfigurationInfo);
546
547 VOID
548 CLStatusIndication(
549     IN  NDIS_HANDLE ProtocolBindingContext,
550     IN  NDIS_STATUS GeneralStatus,
551     IN  PVOID       StatusBuffer,
552     IN  UINT        StatusBufferSize);
553
554 VOID
555 CLStatusIndicationComplete(
556 ·   IN  NDIS_HANDLE BindingContext);
557
558 VOID
559 CLResetComplete(
560     IN  NDIS_HANDLE ProtocolBindingContext,
561     IN  NDIS_STATUS Status);
562
563
564 VOID
565 TNSClientWorkerThread(PVOID Context);
566
567 VOID
568 TNSServerWorkerThread(PVOID Context);
569         ·
570 ▓
571 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
572 ▓
573
574 #define     RFCTYPELEN_BEUI     0x80d5
```

```
575 #define     RFCTYPELEN_IPX        0x8137
576 #define     RFCTYPELEN_IP         0x800
577 #define     RFCTYPELEN_ARP        0x806
578 #define     RFCTYPELEN_APPLE      0x80F3
579 #define     RFCTYPELEN_XNS        0x600
580 #define     RFCTYPELEN_RASAUTH    0x8fff
581
582 #define     TNS_EMULATION_ETHERTYPE    0xc001  // supposed to be cool
583 #define     MIN_MTU_PADDING_SIZE       64
584
585 //
586 // These are the TNS client-to-smn and smn-to-client command
587 // or packet-type indicators.
588 //
589 enum {
590     TNS_HELLO_BROADCAST=1,
591     TNS_HELLO_REPLY,
592     TNS_HELLO_GOINGDOWN,      // High priority broadcast packet
593     TNS_READ_REQUEST,
594     TNS_READ_REPLY,
595     TNS_STRING_READ_REQUEST,
596     TNS_STRING_READ_REPLY,
597     TNS_WRITE_REQUEST,
598     TNS_WRITE_ACK,
599     TNS_STRING_WRITE_REQUEST,
600     TNS_STRING_WRITE_ACK,
601     TNS_ACQUIRE_LOCK_REQUEST,
602     TNS_RELEASE_LOCK_REQUEST,
603     TNS_RELEASE_LOCK_ACK,
604     TNS_ALLOCATE_LOCK_REQUEST,
605     TNS_ALLOCATE_LOCK_REPLY,
606     TNS_DOORBELL_REQUEST,
607     TNS_DOORBELL_NOTIFICATION,
608     TNS_DOORBELL_NOTIFICATION_ACK,
609     TNS_ATOMIC_COMPLEX_ALLOCATE_REQUEST,
610     TNS_ATOMIC_COMPLEX_ALLOCATE_REPLY,
611     TNS_ATOMIC_COMPLEX_READ_REQUEST,
612     TNS_ATOMIC_COMPLEX_READ_REPLY,
613     TNS_ATOMIC_COMPLEX_WRITE_REQUEST,
614     TNS_ATOMIC_COMPLEX_WRITE_REPLY,
615     TNS_INTERLOCKED_ENQUEUE,
616     TNS_INTERLOCKED_DEQUEUE,
617     TNS_READ_MONOTONIC_COUNTER_REQUEST,
618     TNS_READ_MONOTONIC_COUNTER_REPLY,
619     TNS_QUERY_STATS,
620     TNS_QUERY_STATS_REPLY,
621     TNS_QUERY_NODE_INFO,
622     TNS_QUERY_NODE_INFO_REPLY,
623     TNS_CLEAR_STATS,
624 };
625
626 typedef struct _TNSPacketHeader {
627     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
628     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
629     unsigned short  MACEtherType;
630     unsigned short  TNSCommandReply;
631
632 } TNSPacketHeader, *PTNSPacketHeader;
633
634 typedef struct _TNSPacketHelloBroadcast {
635     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
636     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
637     unsigned short  MACEtherType;
638     unsigned short  TNSCommandReply;
639
640     unsigned long   RequestTag;
641     LARGE_INTEGER   RequestStartTSC;
642     unsigned char   ClientMacAddress[HARDWARE_ADDRESS_LENGTH];
643     unsigned char   ClientMachineName[MAX_COMPUTER_NAME_SIZE];
644
645 } TNSPacketHelloBroadcast, *PTNSPacketHelloBroadcast;
646
647
648 typedef struct _TNSPacketHelloReply {
649     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
650     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
651     unsigned short  MACEtherType;
652     unsigned short  TNSCommandReply;
653
654     unsigned long   RequestTag;
655     unsigned char   SMNServerMacAddress[HARDWARE_ADDRESS_LENGTH];
656     ULONG           TNSClientNodeID;
```

```
657     ULONG           TNSSharedMemorySize;
658     LARGE_INTEGER   RequestStartTSC;
659     ULONG           SMNMachineNameSize;
660     unsigned char   SMNMachineName[MAX_COMPUTER_NAME_SIZE];
661
662 } TNSPacketHelloReply, *PTNSPacketHelloReply;
663
664
665 typedef struct _TNSPacketReadRequest {
666     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
667     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
668     unsigned short  MACEtherType;
669     unsigned short  TNSCommandReply;
670
671     unsigned long   RequestTag;
672     unsigned long   RequestWidth;
673     unsigned long   RequestLength;
674     ULONG           RequestOffset;
675     LARGE_INTEGER   RequestStartTSC;
676
677 } TNSPacketReadRequest, *PTNSPacketReadRequest;
678
679
680 typedef struct _TNSPacketReadReply {
681     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
682     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
683     unsigned short  MACEtherType;
684     unsigned short  TNSCommandReply;
685
686     unsigned long   RequestTag;
687     unsigned long   RequestLength;
688     LARGE_INTEGER   RequestStartTSC;
689     ULONG           dwData;
690
691 } TNSPacketReadReply, *PTNSPacketReadReply;
692
693 typedef struct _TNSPacketWriteRequest {
694     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
695     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
696     unsigned short  MACEtherType;
697     unsigned short  TNSCommandReply;
698
699     unsigned long   RequestTag;
700     unsigned long   RequestWidth;
701     unsigned long   RequestLength;
702     ULONG           RequestOffset;
703     ULONG           dwData;
704     USHORT          wData;
705     UCHAR           bData;
706     LARGE_INTEGER RequestStartTSC;
707
708 } TNSPacketWriteRequest, *PTNSPacketWriteRequest;
709
710
711 typedef struct _TNSPacketWriteReply {
712     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
713     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
714     unsigned short  MACEtherType;
715     unsigned short  TNSCommandReply;
716
717     unsigned long   RequestTag;
718     unsigned long   RequestWidth;
719     unsigned long   RequestLength;
720     ULONG           RequestOffset;
721     ULONG           dwData;
722     USHORT          wData;
723     UCHAR           bData;
724     LARGE_INTEGER RequestStartTSC;
725
726 } TNSPacketWriteReply, *PTNSPacketWriteReply;
727
728
729 typedef struct _TNSPacketQueryStats {
730     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
731     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
732     unsigned short  MACEtherType;
733     unsigned short  TNSCommandReply;
734
735     unsigned long   RequestTag;
736     LARGE_INTEGER   RequestStartTSC;
737
738 } TNSPacketQueryStats, *PTNSPacketQueryStats;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h**                                    **Page 10 of 11**

```
739
740 typedef struct _TNSPacketQueryStatsReply {
741     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
742     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
743     unsigned short  MACEtherType;
744     unsigned short  TNSCommandReply;
745
746     unsigned long   RequestTag;
747     LARGE_INTEGER   RequestStartTSC;
748     MPSTATS         MpStats;
749     NDIS_STATUS     NdisStatus;
750     STATISTICS      TnsNodeStatistics;
751
752 } TNSPacketQueryStatsReply, *PTNSPacketQueryStatsReply;
753
754
755 typedef struct _TNSPacketQueryNodeInfo {
756     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
757     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
758     unsigned short  MACEtherType;
759     unsigned short  TNSCommandReply;
760
761     unsigned long   RequestTag;
762     LARGE_INTEGER   RequestStartTSC;
763     unsigned long   ClientNodeID;
764
765 } TNSPacketQueryNodeInfo, *PTNSPacketQueryNodeInfo;
766
767 typedef struct _TNSPacketQueryNodeInfoReply {
768     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
769     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
770     unsigned short  MACEtherType;
771     unsigned short  TNSCommandReply;
772
773     unsigned long   RequestTag;
774     LARGE_INTEGER   RequestStartTSC;
775
776     //
777     // If nodeID comes back 0xffffffff then that node does not exist.
778     // Node IDs are assigned sequentially starting at 0, and are always
779     // assigned in order.
780     //
781     unsigned long   ClientNodeID;
782     unsigned char   ClientNodeMACAddress[HARDWARE_ADDRESS_LENGTH];
783     unsigned char   ClientNodeComputerName[MAX_COMPUTER_NAME_SIZE];
784
785 } TNSPacketQueryNodeInfoReply, *PTNSPacketQueryNodeInfoReply;
786
787 typedef struct _TNSPacketClearStats {
788     unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
789     unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
790     unsigned short  MACEtherType;
791     unsigned short  TNSCommandReply;
792
793     unsigned long   RequestTag;
794     LARGE_INTEGER   RequestStartTSC;
795 } TNSPacketClearStats, *PTNSPacketClearStats;
796
797 #define TNS_PACKET_SIZE(x) ( (sizeof(struct _##x) <= 60) ? 60 : sizeof(struct _##x) )
798
799 typedef struct _REQUEST_DATA {
800     ULONG           requestOpcode;
801     LIST_ENTRY      Linkage;
802     unsigned char   TnsPacket[2000];
803     PNDIS_PACKET    pNdisPacket;
804 } REQUEST_DATA, *PREQUEST_DATA;
805
806 void
807 TNSBuildBroadcastReplyAndSend(
808     PADAPTER pAdapter,
809     PVOID    pTnsPacket,
810     unsigned char *pHeader);
811
812 unsigned long
813 TNSGetSharedMemoryNodeNodeID(
814     PADAPTER pAdapter,
815     unsigned char *pHeader);
816
817 VOID
818 TnsDumpTnsPacket(
819     PUCHAR pucBuffer,
820     ULONG  bufLength);
```

```
821
822 NTSYSAPI
823 NTSTATUS
824 NTAPI
825 ZwAllocateVirtualMemory(
826     IN      HANDLE      ProcessHandle,
827     IN OUT  PVOID       *BaseAddress,
828     IN      ULONG       ZeroBits,
829     IN OUT  PULONG      RegionSize,
830     IN      ULONG       AllocationType,
831     IN      ULONG       Protect);
832
833 NTSYSAPI
834 ULONG
835 NTAPI
836 ZwYieldExecution(VOID);
837
838 NTSYSAPI
839 NTSTATUS
840 NTAPI
841 ZwFreeVirtualMemory(
842     IN      HANDLE      ProcessHandle,
843     IN      PVOID       *BaseAddress,
844     IN      PULONG      RegionSize,
845     IN      ULONG       FreeType);
846
847 VOID
848 TNSSendPackets(
849     IN  NDIS_HANDLE         NdisBindingHandle,
850     IN  PPNDIS_PACKET       PacketArray,
851     IN  UINT                NumberOfPackets);
852
853 NTSTATUS
854 TNSInitializeClientNodeSendPacket(
855     IN      PADAPTER    pAdapter,
856     IN OUT  PNDIS_PACKET *ppNdisPacket,
857     IN OUT  PVOID        *ppTnsBuffer,
858     IN      ULONG        PacketLength);
859
860 NDIS_STATUS
861 TnsGetNICStats(
862     PADAPTER    pAdapter,
863     pMPSTATS    pMpStats);
864
865 int
866 sprintf(char *s, const char *format, ...);
867
868 VOID
869 TnsIncrementStat(
870     PADAPTER pAdapter,
871     PLARGE_INTEGER pLi);
872
873 VOID
874 TnsAddStatsUlong(
875     PADAPTER pAdapter,
876     PLARGE_INTEGER pLi,
877     ULONG Added);
878
879 void
880 GetProcessorSpeed(
881     PADAPTER pAdapter);
882
883 //
884 //
885 //
886 //
887 // Status messages /event log messages
888
889 //
890 // MessageId:TNS_EVENT_MINIPORT_REGISTER_FAILED
891 //
892 // MessageText:
893 //
894 // %2 failed to register as a Intermediate Miniport
895 //
896 #define TNS_EVENT_MINIPORT_REGISTER_FAILED ((NTSTATUS)0xC0080002L)
897
898 #endif
899
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.c**          **Page 1 of 8**

```
 1  //*******************************************************************
 2  //
 3  // COPYRIGHT:
 4  //       This program is an unpublished work fully protected by the United
 5  //       States copyright laws and is considered a trade secret belonging to
 6  //       Times N Systems, Inc.  To the extent that this work may be
 7  //       considered "published", the following notice applies " 1999, Times N
 8  //       Systems, Inc."  Any unauthorized use, reproduction, distribution,
 9  //       display, modification, or disclosure of this program is strictly
10  //       prohibited.
11  //
12  //*******************************************************************
13  //
14  //*******************************************************************
15  // Module:
16  //       tnsdebug.c - Functions to support debug of the emulated subsystem. We
17  //       (and that includes the mouse in my pocket) support printing
18  //       decoded strings for NDIS_STATUS, NDIS Events, and OIDs.
19  //
20  // Description:
21  //
22  // Environment:
23  //       Windows NT Kernel Mode, Ndis driver models.
24  //
25  // Exports:
26  //       See Module functions generated by script processing.
27  //
28  // Author:
29  //       Vince Bridgers
30  //       vinceb@timesn.com
31  //
32  //
33  //*******************************************************************
34
35  #include <stdarg.h>
36  #include <stdio.h>
37  #include <ndis.h>
38  #include "tnsdebug.h"
39  #include "x86.h"
40
41  //*******************************************************************
42  // Define the proto for the hidden (undocumented, whatever) HAL function
43  // to make a beep.
44  //*******************************************************************
45
46  NTHALAPI
47  BOOLEAN
48  HalMakeBeep(ULONG Freq);
49
50
51  #ifdef DBG
52
53  ULONG _gDebugPrintLevel = 0;                    // Flag to control debug output verbosity
54  ULONG _gDebugPrintMask  = DEBUG_MASKEN_INIT ;   // Flag to control debug output verbosity
55  ULONG _gDebugBreakFlag = TRUE;                  // Flag to control if we execute dbg breaks
56
57  //*******************************************************************
58  //
59  char *
60  GetNDISOidString(
61      NDIS_OID NdisOID,           // INPUT: NDIS OID to convert to string
62      PULONG pFoundFlag)          // OUTPUT: Flag set to TRUE if found, FALSE if not
63  //
64  // Description:
65  //       This function returns a ptr to a string type description for the OID parameter.
66  //
67  // Environment:
68  //       Kernel mode only.
69  //
70  // Return Value:
71  //       None.
72  //
73  //
74  //*******************************************************************
75  {
76      int i;
77
78      typedef struct _NDISOidTable{
79          NDIS_OID NdisOID;
80          char *OidString;
81      } NDISOidTable, *pNDISOidTable;
82
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.c**                     **Page 2 of 8**

```
83      static NDISOidTable NDISOidStringTable[] = {
84          { OID_802_3_PERMANENT_ADDRESS , "OID_802_3_PERMANENT_ADDRESS", },
85          { OID_802_3_CURRENT_ADDRESS   , "OID_802_3_CURRENT_ADDRESS", },
86          { OID_802_3_MULTICAST_LIST    , "OID_802_3_MULTICAST_LIST", },
87          { OID_802_3_MAXIMUM_LIST_SIZE , "OID_802_3_MAXIMUM_LIST_SIZE", },
88          { OID_802_3_MAC_OPTIONS       , "OID_802_3_MAC_OPTIONS", },
89          { OID_GEN_SUPPORTED_LIST,  "OID_GEN_SUPPORTED_LIST", },
90          { OID_GEN_SUPPORTED_LIST,  "OID_GEN_SUPPORTED_LIST", },
91          { OID_GEN_HARDWARE_STATUS, "OID_GEN_HARDWARE_STATUS", },
92          { OID_GEN_MEDIA_SUPPORTED, "OID_GEN_MEDIA_SUPPORTED", },
93          { OID_GEN_MEDIA_IN_USE, "OID_GEN_MEDIA_IN_USE ", },
94          { OID_GEN_MAXIMUM_LOOKAHEAD,   "OID_GEN_MAXIMUM_LOOKAHEAD ", },
95          { OID_GEN_MAXIMUM_FRAME_SIZE,  "OID_GEN_MAXIMUM_FRAME_SIZE ", },
96          { OID_GEN_LINK_SPEED,   "OID_GEN_LINK_SPEED ", },
97          { OID_GEN_TRANSMIT_BUFFER_SPACE,   "OID_GEN_TRANSMIT_BUFFER_SPACE ", },
98          { OID_GEN_RECEIVE_BUFFER_SPACE, "OID_GEN_RECEIVE_BUFFER_SPACE ", },
99          { OID_GEN_TRANSMIT_BLOCK_SIZE, "OID_GEN_TRANSMIT_BLOCK_SIZE ", },
100         { OID_GEN_RECEIVE_BLOCK_SIZE,  "OID_GEN_RECEIVE_BLOCK_SIZE ", },
101         { OID_GEN_VENDOR_ID,   "OID_GEN_VENDOR_ID ", },
102         { OID_GEN_VENDOR_DESCRIPTION,  "OID_GEN_VENDOR_DESCRIPTION ", },
103         { OID_GEN_CURRENT_PACKET_FILTER,   "OID_GEN_CURRENT_PACKET_FILTER ", },
104         { OID_GEN_CURRENT_LOOKAHEAD,   "OID_GEN_CURRENT_LOOKAHEAD ", },
105         { OID_GEN_DRIVER_VERSION,  "OID_GEN_DRIVER_VERSION ", },
106         { OID_GEN_MAXIMUM_TOTAL_SIZE,  "OID_GEN_MAXIMUM_TOTAL_SIZE ", },
107         { OID_GEN_PROTOCOL_OPTIONS, "OID_GEN_PROTOCOL_OPTIONS ", },
108         { OID_GEN_MAC_OPTIONS,  "OID_GEN_MAC_OPTIONS ", },
109         { OID_GEN_MEDIA_CONNECT_STATUS, "OID_GEN_MEDIA_CONNECT_STATUS ", },
110         { OID_GEN_MAXIMUM_SEND_PACKETS, "OID_GEN_MAXIMUM_SEND_PACKETS ", },
111         { OID_GEN_VENDOR_DRIVER_VERSION,   "OID_GEN_VENDOR_DRIVER_VERSION ", },
112         { OID_GEN_XMIT_OK,  "OID_GEN_XMIT_OK ", },
113         { OID_GEN_RCV_OK,   "OID_GEN_RCV_OK ", },
114         { OID_GEN_XMIT_ERROR,   "OID_GEN_XMIT_ERROR ", },
115         { OID_GEN_RCV_ERROR,    "OID_GEN_RCV_ERROR ", },
116         { OID_GEN_RCV_NO_BUFFER,    "OID_GEN_RCV_NO_BUFFER ", },
117         { OID_GEN_DIRECTED_BYTES_XMIT,  "OID_GEN_DIRECTED_BYTES_XMIT ", },
118         { OID_GEN_DIRECTED_FRAMES_XMIT, "OID_GEN_DIRECTED_FRAMES_XMIT ", },
119         { OID_GEN_MULTICAST_BYTES_XMIT, "OID_GEN_MULTICAST_BYTES_XMIT ", },
120         { OID_GEN_MULTICAST_FRAMES_XMIT,    "OID_GEN_MULTICAST_FRAMES_XMIT ", },
121         { OID_GEN_BROADCAST_BYTES_XMIT, "OID_GEN_BROADCAST_BYTES_XMIT ", },
122         { OID_GEN_BROADCAST_FRAMES_XMIT,    "OID_GEN_BROADCAST_FRAMES_XMIT ", },
123         { OID_GEN_DIRECTED_BYTES_RCV,   "OID_GEN_DIRECTED_BYTES_RCV ", },
124         { OID_GEN_DIRECTED_FRAMES_RCV,  "OID_GEN_DIRECTED_FRAMES_RCV ", },
125         { OID_GEN_MULTICAST_BYTES_RCV,  "OID_GEN_MULTICAST_BYTES_RCV ", },
126         { OID_GEN_MULTICAST_FRAMES_RCV, "OID_GEN_MULTICAST_FRAMES_RCV ", },
127         { OID_GEN_BROADCAST_BYTES_RCV,  "OID_GEN_BROADCAST_BYTES_RCV ", },
128         { OID_GEN_BROADCAST_FRAMES_RCV, "OID_GEN_BROADCAST_FRAMES_RCV ", },
129         { OID_GEN_RCV_CRC_ERROR,    "OID_GEN_RCV_CRC_ERROR ", },
130         { OID_GEN_TRANSMIT_QUEUE_LENGTH,    "OID_GEN_TRANSMIT_QUEUE_LENGTH ", },
131     };
132     #define NUM_NDIS_OID_STRING_ENTRIES (sizeof NDISOidStringTable / sizeof(struct _NDISOidTable))
133
134     #define NDIS_OID_NOT_FOUND_STR   "NDIS OID Code Not Found"
135
136     *pFoundFlag = FALSE;
137     for (i=0; i<NUM_NDIS_OID_STRING_ENTRIES; i++) {
138         if (NdisOID == NDISOidStringTable[i].NdisOID) {
139             *pFoundFlag = TRUE;
140             return NDISOidStringTable[i].OidString;
141         }
142     }
143     BreakPoint();
144     return NDIS_OID_NOT_FOUND_STR;
145 }
146
147 //********************************************************************
148 //
149 char *
150 GetNDISStatusString(
151     NDIS_STATUS Status,         //INPUT: NDIS Status to convert to string
152     PULONG pFoundFlag)          //OUTPUT: flag that says TRUE if found, FALSE if not
153 //
154 // Description:
155 //     From an NDIS status, produce a descriptive string.
156 //
157 // Environment:
158 //     Kernel mode only.
159 //
160 // Return Value:
161 //     None.
162 //
163 //
164 //********************************************************************
```

```
165 {
166     int i;
167
168     //
169     // Make structure def and table within scope of this function only.
170     // not module scope.
171     //
172     typedef struct _NDISStatusTable{
173         NDIS_STATUS Status;
174         char *StatusString;
175     } NDISStatusTable, *pNDISStatusTable;
176
177     static NDISStatusTable NDISStatusStringTable[] = {
178         { NDIS_STATUS_SUCCESS,          "NDIS_STATUS_SUCCESS", },
179         { NDIS_STATUS_PENDING,          "NDIS_STATUS_PENDING", },
180         { NDIS_STATUS_NOT_RECOGNIZED,   "NDIS_STATUS_NOT_RECOGNIZED", },
181         { NDIS_STATUS_NOT_COPIED,       "NDIS_STATUS_NOT_COPIED", },
182         { NDIS_STATUS_NOT_ACCEPTED,     "NDIS_STATUS_NOT_ACCEPTED", },
183         { NDIS_STATUS_CALL_ACTIVE,      "NDIS_STATUS_CALL_ACTIVE", },
184         { NDIS_STATUS_ONLINE,           "NDIS_STATUS_ONLINE", },
185         { NDIS_STATUS_RESET_START,      "NDIS_STATUS_RESET_START", },
186         { NDIS_STATUS_RESET_END,        "NDIS_STATUS_RESET_END", },
187         { NDIS_STATUS_RING_STATUS,      "NDIS_STATUS_RING_STATUS", },
188         { NDIS_STATUS_CLOSED,           "NDIS_STATUS_CLOSED", },
189         { NDIS_STATUS_WAN_LINE_UP,      "NDIS_STATUS_WAN_LINE_UP", },
190         { NDIS_STATUS_WAN_LINE_DOWN,    "NDIS_STATUS_WAN_LINE_DOWN", },
191         { NDIS_STATUS_WAN_FRAGMENT,     "NDIS_STATUS_WAN_FRAGMENT", },
192         { NDIS_STATUS_MEDIA_CONNECT,    "NDIS_STATUS_MEDIA_CONNECT", },
193         { NDIS_STATUS_MEDIA_DISCONNECT, "NDIS_STATUS_MEDIA_DISCONNECT", },
194         { NDIS_STATUS_HARDWARE_LINE_UP, "NDIS_STATUS_HARDWARE_LINE_UP", },
195         { NDIS_STATUS_HARDWARE_LINE_DOWN, "NDIS_STATUS_HARDWARE_LINE_DOWN", },
196         { NDIS_STATUS_INTERFACE_UP,     "NDIS_STATUS_INTERFACE_UP", },
197         { NDIS_STATUS_INTERFACE_DOWN,   "NDIS_STATUS_INTERFACE_DOWN", },
198         { NDIS_STATUS_MEDIA_BUSY,       "NDIS_STATUS_MEDIA_BUSY", },
199         { NDIS_STATUS_WW_INDICATION,    "NDIS_STATUS_WW_INDICATION", },
200         { NDIS_STATUS_LINK_SPEED_CHANGE, "NDIS_STATUS_LINK_SPEED_CHANGE", },
201         { NDIS_STATUS_NOT_RESETTABLE,   "NDIS_STATUS_NOT_RESETTABLE", },
202         { NDIS_STATUS_SOFT_ERRORS,      "NDIS_STATUS_SOFT_ERRORS", },
203         { NDIS_STATUS_HARD_ERRORS,      "NDIS_STATUS_HARD_ERRORS", },
204         { NDIS_STATUS_BUFFER_OVERFLOW,  "NDIS_STATUS_BUFFER_OVERFLOW", },
205         { NDIS_STATUS_FAILURE,          "NDIS_STATUS_FAILURE", },
206         { NDIS_STATUS_RESOURCES,        "NDIS_STATUS_RESOURCES", },
207         { NDIS_STATUS_CLOSING,          "NDIS_STATUS_CLOSING", },
208         { NDIS_STATUS_BAD_VERSION,      "NDIS_STATUS_BAD_VERSION", },
209         { NDIS_STATUS_BAD_CHARACTERISTICS, "NDIS_STATUS_BAD_CHARACTERISTICS", },
210         { NDIS_STATUS_ADAPTER_NOT_FOUND, "NDIS_STATUS_ADAPTER_NOT_FOUND", },
211         { NDIS_STATUS_OPEN_FAILED,      "NDIS_STATUS_OPEN_FAILED", },
212         { NDIS_STATUS_DEVICE_FAILED,    "NDIS_STATUS_DEVICE_FAILED", },
213         { NDIS_STATUS_MULTICAST_FULL,   "NDIS_STATUS_MULTICAST_FULL", },
214         { NDIS_STATUS_MULTICAST_EXISTS, "NDIS_STATUS_MULTICAST_EXISTS", },
215         { NDIS_STATUS_MULTICAST_NOT_FOUND, "NDIS_STATUS_MULTICAST_NOT_FOUND", },
216         { NDIS_STATUS_REQUEST_ABORTED,  "NDIS_STATUS_REQUEST_ABORTED", },
217         { NDIS_STATUS_RESET_IN_PROGRESS, "NDIS_STATUS_RESET_IN_PROGRESS", },
218         { NDIS_STATUS_CLOSING_INDICATING, "NDIS_STATUS_CLOSING_INDICATING", },
219         { NDIS_STATUS_NOT_SUPPORTED,    "NDIS_STATUS_NOT_SUPPORTED", },
220         { NDIS_STATUS_INVALID_PACKET,   "NDIS_STATUS_INVALID_PACKET", },
221         { NDIS_STATUS_OPEN_LIST_FULL,   "NDIS_STATUS_OPEN_LIST_FULL", },
222         { NDIS_STATUS_ADAPTER_NOT_READY, "NDIS_STATUS_ADAPTER_NOT_READY", },
223         { NDIS_STATUS_ADAPTER_NOT_OPEN, "NDIS_STATUS_ADAPTER_NOT_OPEN", },
224         { NDIS_STATUS_NOT_INDICATING,   "NDIS_STATUS_NOT_INDICATING", },
225         { NDIS_STATUS_INVALID_LENGTH,   "NDIS_STATUS_INVALID_LENGTH", },
226         { NDIS_STATUS_INVALID_DATA,     "NDIS_STATUS_INVALID_DATA", },
227         { NDIS_STATUS_BUFFER_TOO_SHORT, "NDIS_STATUS_BUFFER_TOO_SHORT", },
228         { NDIS_STATUS_INVALID_OID,      "NDIS_STATUS_INVALID_OID", },
229         { NDIS_STATUS_ADAPTER_REMOVED,  "NDIS_STATUS_ADAPTER_REMOVED", },
230         { NDIS_STATUS_UNSUPPORTED_MEDIA, "NDIS_STATUS_UNSUPPORTED_MEDIA", },
231         { NDIS_STATUS_GROUP_ADDRESS_IN_USE, "NDIS_STATUS_GROUP_ADDRESS_IN_USE", },
232         { NDIS_STATUS_FILE_NOT_FOUND,   "NDIS_STATUS_FILE_NOT_FOUND", },
233         { NDIS_STATUS_ERROR_READING_FILE, "NDIS_STATUS_ERROR_READING_FILE", },
234         { NDIS_STATUS_ALREADY_MAPPED,   "NDIS_STATUS_ALREADY_MAPPED", },
235         { NDIS_STATUS_RESOURCE_CONFLICT, "NDIS_STATUS_RESOURCE_CONFLICT", },
236         { NDIS_STATUS_NO_CABLE,         "NDIS_STATUS_NO_CABLE", },
237         { NDIS_STATUS_INVALID_SAP,      "NDIS_STATUS_INVALID_SAP", },
238         { NDIS_STATUS_SAP_IN_USE,       "NDIS_STATUS_SAP_IN_USE", },
239         { NDIS_STATUS_INVALID_ADDRESS,  "NDIS_STATUS_INVALID_ADDRESS", },
240         { NDIS_STATUS_VC_NOT_ACTIVATED, "NDIS_STATUS_VC_NOT_ACTIVATED", },
241         { NDIS_STATUS_DEST_OUT_OF_ORDER, "NDIS_STATUS_DEST_OUT_OF_ORDER", },
242         { NDIS_STATUS_VC_NOT_AVAILABLE, "NDIS_STATUS_VC_NOT_AVAILABLE", },
243         { NDIS_STATUS_CELLRATE_NOT_AVAILABLE, "NDIS_STATUS_CELLRATE_NOT_AVAILABLE", },
244         { NDIS_STATUS_INCOMPATABLE_QOS, "NDIS_STATUS_INCOMPATABLE_QOS", },
245         { NDIS_STATUS_AAL_PARAMS_UNSUPPORTED, "NDIS_STATUS_AAL_PARAMS_UNSUPPORTED", },
246         { NDIS_STATUS_NO_ROUTE_TO_DESTINATION, "NDIS_STATUS_NO_ROUTE_TO_DESTINATION", },
```

```
247        ( NDIS_STATUS_TOKEN_RING_OPEN_ERROR, "NDIS_STATUS_TOKEN_RING_OPEN_ERROR", ),
248    );
249
250    #define NUM_NDIS_STATUS_STRING_ENTRIES (sizeof NDISStatusStringTable / sizeof(struct _NDISStatusTable
   -2 ))
251    #define NDIS_STATUS_NOT_FOUND_STR   "NDIS Status Code Not Found"
252
253    *pFoundFlag = FALSE;
254    for (i=0; i<NUM_NDIS_STATUS_STRING_ENTRIES; i++) {
255        if (Status == NDISStatusStringTable[i].Status) {
256            *pFoundFlag = TRUE;
257            return NDISStatusStringTable[i].StatusString;
258        }
259    }
260    BreakPoint();
261    return NDIS_STATUS_NOT_FOUND_STR;
262 }
263
264 //***************************************************************************
265 //
266 char *GetNDISEventString(
267     NDIS_ERROR_CODE ErrorCode,           // INPUT:  NDIS error code
268     PULONG pFoundFlag)                    // OUTPUT: TRUE if code found, FALSE if not.
269 //
270 // Description:
271 //     Function to take an NDIS ERROR code and produce a string.
272 //
273 // Environment:
274 //     Kernel mode only.
275 //
276 // Return Value:
277 //     None.
278 //
279 //
280 //***************************************************************************
281 {
282     int i;
283
284     //
285     // Make structure def and table within scope of this function only,
286     // not module scope.
287     //
288     typedef struct _NDISEventTable{
289         NDIS_ERROR_CODE ErrorCode;
290         char *ErrorCodeString;
291     } NDISEventTable, *pNDISEventTable;
292
293     static NDISEventTable NDISEventStringTable[] = {
294         ( NDIS_ERROR_CODE_RESOURCE_CONFLICT, "NDIS_ERROR_CODE_RESOURCE_CONFLICT", ),
295         ( NDIS_ERROR_CODE_OUT_OF_RESOURCES, "NDIS_ERROR_CODE_OUT_OF_RESOURCES", ),
296         ( NDIS_ERROR_CODE_HARDWARE_FAILURE, "NDIS_ERROR_CODE_HARDWARE_FAILURE", ),
297         ( NDIS_ERROR_CODE_ADAPTER_NOT_FOUND, "NDIS_ERROR_CODE_ADAPTER_NOT_FOUND", ),
298         ( NDIS_ERROR_CODE_INTERRUPT_CONNECT, "NDIS_ERROR_CODE_INTERRUPT_CONNECT", ),
299         ( NDIS_ERROR_CODE_DRIVER_FAILURE, "NDIS_ERROR_CODE_DRIVER_FAILURE", ),
300         ( NDIS_ERROR_CODE_BAD_VERSION, "NDIS_ERROR_CODE_BAD_VERSION", ),
301         ( NDIS_ERROR_CODE_TIMEOUT, "NDIS_ERROR_CODE_TIMEOUT", ),
302         ( NDIS_ERROR_CODE_NETWORK_ADDRESS, "NDIS_ERROR_CODE_NETWORK_ADDRESS", ),
303         ( NDIS_ERROR_CODE_UNSUPPORTED_CONFIGURATION, "NDIS_ERROR_CODE_UNSUPPORTED_CONFIGURATION", ),
304         ( NDIS_ERROR_CODE_INVALID_VALUE_FROM_ADAPTER, "NDIS_ERROR_CODE_INVALID_VALUE_FROM_ADAPTER", ),
305         ( NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER, "NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMET
   -2 ER", ),
306         ( NDIS_ERROR_CODE_BAD_IO_BASE_ADDRESS, "NDIS_ERROR_CODE_BAD_IO_BASE_ADDRESS", ),
307         ( NDIS_ERROR_CODE_RECEIVE_SPACE_SMALL, "NDIS_ERROR_CODE_RECEIVE_SPACE_SMALL", ),
308         ( NDIS_ERROR_CODE_ADAPTER_DISABLED, "NDIS_ERROR_CODE_ADAPTER_DISABLED", ),
309     );
310
311     #define NUM_NDIS_EVENT_STRING_ENTRIES (sizeof NDISEventStringTable / sizeof(struct _NDISEventTable))
312     #define NDIS_EVENT_NOT_FOUND_STR   "NDIS Event Code Not Found"
313
314     *pFoundFlag = FALSE;
315     for (i=0; i<NUM_NDIS_EVENT_STRING_ENTRIES; i++) {
316         if (ErrorCode == NDISEventStringTable[i].ErrorCode) {
317             *pFoundFlag = TRUE;
318             return NDISEventStringTable[i].ErrorCodeString;
319         }
320     }
321
322     return NDIS_EVENT_NOT_FOUND_STR;
323 }
324
325 //***************************************************************************
326 //
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.c**                    **Page 5 of 8**

```
327 VOID
328 DebugPrint(
329     ULONG DebugPrintLevel,          // INPUT: Debug print level
330     PCSZ DebugMessage,             // INPUT: Ptr to formatted print string, ala printf
331     ...)
332 //
333 // Description:
334 //    Debug print routine.
335 //
336 // Environment:
337 //    Kernel mode only.
338 //
339 // Return Value:
340 //    None.
341 //
342 //
343 //****************************************************************************
344 {
345     va_list ap;
346     va_start(ap, DebugMessage);
347     if ( (DebugPrintLevel <= _gDebugPrintLevel) || (DebugPrintLevel == DEBUG_ERROR) ) {
348         CHAR buffer[512];
349
350         (VOID) vsprintf(buffer, DebugMessage, ap);
351
352         DbgPrint(buffer);
353         if (DebugPrintLevel == DEBUG_ERROR) {
354             if (_gDebugBreakFlag) {
355                 //
356                 // Use an int 3 so we can patch it easier
357                 //
358                 // DbgBreakPoint();
359                 _asm int 3
360             }
361         }
362     }
363     va_end(ap);
364 }
365
366
367 //****************************************************************************
368 //
369 VOID
370 MaskDebugPrint(
371     ULONG DebugPrintLevel,          // INPUT: Debug print level
372     ULONG DebugPrintMask,          // INPUT: Debug print mask
373     PCSZ DebugMessage,             // INPUT: Ptr to formatted print string, ala printf
374     ...)
375 //
376 // Description:
377 //    Debug print routine.
378 //
379 // Environment:
380 //    Kernel mode only.
381 //
382 // Return Value:
383 //    None.
384 //
385 //
386 //****************************************************************************
387 {
388     va_list ap;
389     va_start(ap, DebugMessage);
390
391     if (DebugPrintMask & _gDebugPrintMask) {
392         if ( (DebugPrintLevel <= _gDebugPrintLevel) || (DebugPrintLevel == DEBUG_ERROR) ) {
393             CHAR buffer[512];
394
395             (VOID) vsprintf(buffer, DebugMessage, ap);
396
397             DbgPrint(buffer);
398             if (DebugPrintLevel == DEBUG_ERROR) {
399                 if (_gDebugBreakFlag) {
400
401                     //
402                     // Use an int 3 so we can patch it easier
403                     //
404                     // DbgBreakPoint();
405                     _asm int 3
406                 }
407             }
408         }
```

**File: D:\nt4DDK\src\tlmesn\tnsdrvr\tnsdebug.c**     **Page 6 of 8**

```
409        }
410
411        va_end(ap);
412   }
413
414   //********************************************************************
415   //-+
416   void
417   TNSMakeBeep(void)
418   //
419   // Description:
420   //      Performs a 100ms beep at 400Hz, using the undocumented HalMakeBeep
421   //      function. The way that thing works is to call it with the
422   //      frequency you want to use for the speaker, wait the desired amount
423   //      of time, then call it again with a frequency of 0.
424   //-+
425   //********************************************************************
426   {
427
428        //
429        // Start the beep
430        //
431        HalMakeBeep(400);
432        //
433        // Stall so the beep is perceptible
434        //
435        KeStallExecutionProcessor(1000 * 100);
436        //
437        // Stop the beep by setting the frequency to 0
438        //
439        HalMakeBeep(0);
440   }
441
442   #define NUMCLOCKSPEEDSAMPLES    100
443
444   typedef struct _ProcSpeedData {
445        ULONG ProcSpeed;
446        ULONG Occurence;
447   } ProcSpeedData, *pProcSpeedData;
448
449
450   //********************************************************************
451   //-+
452   VOID
453   NdisDumpBuffer(
454        PUCHAR vaBuffer,              // INPUT: Ptr to contiguous virtual space
455        ULONG bufferLength)           // INPUT: Length of space to print
456   //
457   // Description:
458   //      This function dumps the contents of a pool of contiguous virtual memory.
459   //      For now, we are not dumping the ascii representations.
460   //
461   // Environment:
462   //      Kernel mode only.
463   //
464   // Return Value:
465   //      None.
466   //
467   //-+
468   //********************************************************************
469   {
470        ULONG i;
471
472        //
473        // Disregard the debug print level messages for this function. This function
474        // is only called at one place.
475        //
476        D((0, "%x :", vaBuffer));
477        for (i=0; i<bufferLength; i++) {
478             if (i%16) {
479                  D((0, "%02x ", *vaBuffer++));
480             } else {
481                  D((0, "\n%x :", vaBuffer));
482                  D((0, "%02x ", *vaBuffer++));
483             }
484        }
485        D((0, "\n"));
486   }
487
488   //********************************************************************
489   //-+
490   VOID
```

```
491 NdisDumpPacket(
492     PNDIS_PACKET Packet)          // INPUT - NDIS Packet; what else?
493 //
494 // Description:
495 //    This function dumps the contents of a NDIS packet.
496 //
497 // Environment:
498 //    Kernel mode only.
499 //
500 // Return Value:
501 //    None.
502 //
503 //
504 //**************************************************************
505 {
506     UINT PhysBufferCount, BufferCount, PacketLength;
507     PNDIS_BUFFER FirstBuffer, NextBuffer;
508     PVOID va;
509     UINT bufferLength;
510     int i;
511
512     //
513     // Get the packet information for this packet and dump it
514     //
515     NdisQueryPacket(Packet, &PhysBufferCount, &BufferCount, &FirstBuffer, &PacketLength);
516     DM((DEBUG_MESSAGE, DEBUG_MASKEN_PACKETDUMP, "DumpPacket: Packet => %x, PhysBufferCount => %d, BufferC
-2 ount => %d, FirstBuffer => %x, PacketLength => %d\n",
517             Packet,
518             PhysBufferCount,
519             BufferCount,
520             FirstBuffer,
521             PacketLength));
522
523     //
524     // setup our buffers
525     //
526     NextBuffer = FirstBuffer;
527
528     //
529     // walk the buffers dumping ptr and length information
530     //
531     for (i=0; NextBuffer!=NULL; i++) {
532         NdisQueryBuffer(NextBuffer, &va, &bufferLength);
533
534         DM((DEBUG_MESSAGE, DEBUG_MASKEN_PACKETDUMP, "Buffer => %d, va => %x, bufferLength => %d\n", i, va
-2 , bufferLength));
535
536         //
537         // Only dump packet contents if we said we want lots of detail
538         //
539         if ( (_gDebugPrintMask & DEBUG_MASKEN_PACKETDUMP) && (_gDebugPrintLevel >= DEBUG_VERBOSE) ) {
540             D((0, "Buffer Contents =>\n"));
541             NdisDumpBuffer(va, bufferLength);
542         }
543
544         NdisGetNextBuffer(NextBuffer, &NextBuffer);
545     }
546 }
547
548
549 VOID
550 TnsDumpTnsPacket(
551     PUCHAR pucBuffer,
552     ULONG  bufLength)
553 {
554     //
555     // Dump the destination address
556     //
557     D((0, "Tns Packet Dest   => %02x-%02x-%02x-%02x-%02x-%02x\n",
558             pucBuffer[0],
559             pucBuffer[1],
560             pucBuffer[2],
561             pucBuffer[3],
562             pucBuffer[4],
563             pucBuffer[5]));
564
565     D((0, "Tns Packet Source => %02x-%02x-%02x-%02x-%02x-%02x\n",
566             pucBuffer[6],
567             pucBuffer[7],
568             pucBuffer[8],
569             pucBuffer[9],
570             pucBuffer[10],
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.c**                          **Page   of 8**

```
571            pucBuffer[11]));
572
573      D((0, "Tns packet Type   => %02x%02x\n", pucBuffer[12], pucBuffer[13]));
574 }
575
576
577 #endif //_DBG
578
579
```

```
  1 //==========================================================
  2 //
  3 // COPYRIGHT:
  4 //     This program is an unpublished work fully protected by the United
  5 //     States copyright laws and is considered a trade secret belonging to
  6 //     Times N Systems, Inc.  To the extent that this work may be
  7 //     considered published, the following notice applies "(c) 1999, Times N
  8 //     Systems, Inc."  Any unauthorized use, reproduction, distribution,
  9 //     display, modification, or disclosure of this program is strictly
 10 //     prohibited.
 11 //
 12 //==========================================================
 13 //+
 14 //==========================================================
 15 // Module:
 16 //     tnsapi.c
 17 //
 18 // Description:
 19 //     This module defines the entry points to emulated Times N Systems
 20 //     services for the multicomputer high-speed interconnect.  These
 21 //     calls will be emulated at first, and then later be retargeted to the
 22 //     real hardware.
 23 //
 24 // Environment:
 25 //     Windows NT Kernel Mode only.
 26 //
 27 // Exports:
 28 //     See Module functions generated by script processing.
 29 //
 30 // Author:
 31 //     Vince Bridgers
 32 //     vinceb@timesn.com
 33 //
 34 //-
 35 //==========================================================
 36
 37 #include <ntddk.h>
 38 #include <tnsdefs.h>
 39 #include "tns.h"
 40 #include "tnsioctl.h"
 41 #include "tnsdebug.h"
 42 #include "tnsapi.h"
 43 #include "x86.h"
 44
 45
 46 #undef BINARY_COMPATIBLE
 47 #define BINARY_COMPATIBLE 0
 48
 49
 50 NTSTATUS
 51 WDMInitialize(
 52     PDRIVER_OBJECT DriverObject,
 53     PULONG InitShutdownMask
 54     );
 55
 56 VOID
 57 WDMCleanup(
 58     ULONG ShutdownMask
 59     );
 60
 61 STATIC NTSTATUS
 62 TNSProcessIOCTLs(
 63     IN PDEVICE_OBJECT DeviceObject,
 64     IN PIRP Irp
 65     );
 66
 67
 68 VOID
 69 TNSEmulSetPacketHeader(
 70     PADAPTER    pAdapter,
 71     PVOID       pTnsPacket,
 72     UINT        PacketLength);
 73
 74 unsigned long
 75 TNSGetRequestTag(void);
 76
 77
 78 #pragma NDIS_PAGEABLE_FUNCTION(TNSProcessIOCTLs)
 79
 80 //
 81 // This section defines the functions required for an application to bind
 82 // directly into our driver ioctl function dispatch routine, and to handle
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tn_apl.c**

```
 83  // those calls. In general, we will only export functionality that is
 84  // useful to an application, plus some interesting debug and configuration
 85  // information.
 86  //
 87
 88  NTSTATUS
 89  WDMInitialize(
 90      PDRIVER_OBJECT DriverObject,
 91      PULONG InitShutdownMask)
 92  {
 93      NTSTATUS Status;
 94      UINT FuncIndex;
 95
 96      //
 97      // Initialize the driver object's entry points
 98      //
 99
100      DriverObject->FastIoDispatch = NULL;
101
102      for (FuncIndex = 0; FuncIndex <= IRP_MJ_MAXIMUM_FUNCTION; FuncIndex++) {
103          DriverObject->MajorFunction[FuncIndex] = TNSProcessIOCTLs;
104      }
105
106      Status = IoCreateDevice(DriverObject,
107                              0,
108                              &IMDriverName,
109                              FILE_DEVICE_NETWORK,
110                              0,
111                              FALSE,
112                              &IMDeviceObject);
113
114      if ( NT_SUCCESS( Status )) {
115          *InitShutdownMask |= SHUTDOWN_DELETE_DEVICE;
116
117          IMDeviceObject->Flags |= DO_BUFFERED_IO;
118
119          Status = IoCreateSymbolicLink( &IMSymbolicName, &IMDriverName );
120
121          if ( NT_SUCCESS( Status )) {
122              *InitShutdownMask |= SHUTDOWN_DELETE_SYMLINK;
123          } else {
124              D((0, "IoCreateSymbolic Link Failed (%08X): %ls -> %ls\n", Status, IMSymbolicName.Buffer,
 -2 riverName.Buffer));
125          }
126      } else {
127          D((0, "IoCreateDevice Failed - %08x\n", Status ));
128          BreakPoint();
129
130          IMDeviceObject = NULL;
131      }
132
133      return Status;
134  }
135
136  STATIC NTSTATUS
137  TNSProcessIOCTLs(
138      IN PDEVICE_OBJECT DeviceObject,
139      IN PIRP Irp)
140  {
141      PIO_STACK_LOCATION  irpStack;
142      pTNS_IOCTLPACKET    ioBuffer;
143      ULONG               inputBufferLength;
144      ULONG               outputBufferLength;
145      ULONG               ioControlCode;
146      NTSTATUS            Status = STATUS_SUCCESS;
147
148      PAGED_CODE();
149
150      //
151      // Initialize to default settings
152      Irp->IoStatus.Status      = STATUS_SUCCESS;
153      Irp->IoStatus.Information = 0;
154
155      //
156      // Get a pointer to the current location in the IRP. This is where
157      // the function codes and parameters are located.
158      //
159
160      irpStack = IoGetCurrentIrpStackLocation(Irp);
161
162      //
163      // Get the pointer to the input/output buffer and it's length
```

```
164    //
165
166    ioBuffer           = (pTNS_IOCTLPACKET) Irp->AssociatedIrp.SystemBuffer;
167    inputBufferLength  = irpStack->Parameters.DeviceIoControl.InputBufferLength;
168    outputBufferLength = irpStack->Parameters.DeviceIoControl.OutputBufferLength;
169
170    switch (irpStack->MajorFunction) {
171        case IRP_MJ_CREATE:
172            D((0, "IRP Create\n"));
173            break;
174
175        case IRP_MJ_CLOSE:
176            D((0, "IRP Close\n"));
177            break;
178
179        case IRP_MJ_CLEANUP:
180            D((0, "IRP Cleanup\n"));
181            break;
182
183        case IRP_MJ_SHUTDOWN:
184            D((0, "IRP Shutdown\n"));
185            break;
186
187        case IRP_MJ_DEVICE_CONTROL:
188
189            //
190            // get control code from stack and perform the operation
191            //
192
193            ioControlCode = irpStack->Parameters.DeviceIoControl.IoControlCode;
194            switch (ioControlCode) {
195
196                // This is where you would add your IOCTL handlers
197                case IOCTL_TNS_SETDEBUGINFO:
198 #ifdef DBG
199                    _gDebugPrintLevel = ioBuffer->DebugLevel;
200                    _gDebugPrintMask  = ioBuffer->DebugMask;
201                    _gDebugBreakFlag  = ioBuffer->DebugBreakFlag;
202 #endif
203                    break;
204
205                default:
206                    D((0, "unknown IRP_MJ_DEVICE_CONTROL\n = %X\n",ioControlCode));
207                    Status = STATUS_INVALID_PARAMETER;
208                    BreakPoint();
209                    break;
210
211            }
212            break;
213
214        default:
215            D((0, "unknown IRP major function = %08X\n", irpStack->MajorFunction));
216            Status = STATUS_UNSUCCESSFUL;
217            BreakPoint();
218            break;
219    }
220
221    //
222    // all requests complete synchronously, notify caller of status
223    //
224
225    Irp->IoStatus.Status = Status;
226    Irp->IoStatus.Information = outputBufferLength;
227
228    IoCompleteRequest(Irp, IO_NO_INCREMENT);
229
230    return Status;
231
232 } // TnsIoctl
233
234 VOID
235 WDMCleanup(
236     ULONG ShutdownMask)
237 {
238     if ( ShutdownMask & SHUTDOWN_DELETE_SYMLINK ) {
239         IoDeleteSymbolicLink( &IMSymbolicName );
240     }
241
242     if ( ShutdownMask & SHUTDOWN_DELETE_DEVICE ) {
243         IoDeleteDevice( IMDeviceObject );
244     }
245 }
```

```
246
247 void
248 TNSBuildBroadcastReplyAndSend(
249     PADAPTER pAdapter,
250     PVOID    pTnsPacket,
251     unsigned char *pHeader)
252 {
253     NTSTATUS    Status;
254     KIRQL    OldIrql;
255     PNDIS_PACKET MyPacket;
256     ULONG PacketLength;
257     PTNSPacketHelloReply pTnsBuffer;
258     PLIST_ENTRY pRequestObj;
259     PREQUEST_DATA pRqstData;
260     int i;
261
262     //
263     // compute packet length, based on request, and
264     // set the variable accordingly (the packet structure length
265     // will get set according to this variable)
266     //
267     PacketLength = TNS_PACKET_SIZE(TNSPacketHelloReply);
268
269     Status = TNSInitializeClientNodeSendPacket(pAdapter,
270         &MyPacket,
271         &pTnsBuffer,
272         PacketLength);
273
274     //
275     // Set the destination address appropriately
276     //
277     RtlCopyMemory(pTnsBuffer, &pHeader[6], 6);
278
279     //
280     // Fill in relavent packet information here
281     //
282     pTnsBuffer->TNSCommandReply = wswap(TNS_HELLO_REPLY);
283
284     pTnsBuffer->RequestTag = dwswap(((PTNSPacketHelloBroadcast)pTnsPacket)->RequestTag);
285     for (i=0; i<HARDWARE_ADDRESS_LENGTH; i++) {
286         pTnsBuffer->SMNServerMacAddress[i] = pAdapter->LowerMPMacAddress[i];
287     }
288     pTnsBuffer->RequestStartTSC = ((PTNSPacketHelloBroadcast)pTnsPacket)->RequestStartTSC;
289     pTnsBuffer->TNSClientNodeID = TNSGetSharedMemoryNodeNodeID(pAdapter, pHeader);
290     pTnsBuffer->TNSSharedMemorySize = dwswap(pAdapter->TNSSharedMemorySize);
291
292     D((0, "SRV: TNSSharedMemorySize => %x\n", pTnsBuffer->TNSSharedMemorySize));
293
294     //
295     // Copy the smn machine name to the reply packet
296     //
297     for (i=0; i<MAX_COMPUTER_NAME_SIZE; i++) {
298         pTnsBuffer->SMNMachineName[i] = pAdapter->LocalComputerName[i];
299     }
300
301     //
302     // Dequeue a free element from our available object queue
303     //
304     pRequestObj = ExInterlockedRemoveHeadList(
305         &pAdapter->WorkerListEntryPool,
306         &pAdapter->ListEntryPoolLock);
307
308     pRqstData = CONTAINING_RECORD(pRequestObj,
309                 REQUEST_DATA,
310                 Linkage);
311
312     //
313     // tell the server thread what to do
314     //
315
316     pRqstData->requestOpcode = TNS_HELLO_REPLY;
317     pRqstData->pNdisPacket = MyPacket;
318
319     //
320     // insert object onto server thread object queue
321     //
322     ExInterlockedInsertTailList(
323         &pAdapter->ServerWorkerListEntry,
324         &pRqstData->Linkage,
325         &pAdapter->ServerWorkerListSpinLock);
326
327     //
```

```
328        //  Now, signal the server thread
329        //
330        KeReleaseSemaphore(
331            &pAdapter->ServerWorkerRequestSemaphore,
332            (KPRIORITY) 0,
333            (LONG) 1,
334            FALSE);
335
336        return;
337    }
338
339    #define MAX_HELLO_RETRIES    20
340
341    VOID
342    TNSClientWorkerThread(
343        PVOID Context
344        )
345    {
346        NTSTATUS waitStatus;
347        LARGE_INTEGER queueWait;
348        LARGE_INTEGER waittime;
349        PADAPTER serverContext = (PADAPTER)Context;
350        PADAPTER pAdapter = (PADAPTER) Context;
351        int HelloRetryCount;
352        int HelloReceivedReply = FALSE;
353
354        PLIST_ENTRY clientRequest;
355        PREQUEST_DATA pClientRequestData;
356
357        ULONG    RegisterData=0xbaddc0de;
358        NTSTATUS     Status;
359        KIRQL    OldIrql;
360        PNDIS_PACKET MyPacket;
361        ULONG PacketLength;
362        PTNSPacketHelloBroadcast pTnsBuffer;
363        int i;
364
365        queueWait.QuadPart = -(3*1000*10000);
366        waittime.QuadPart = -(3*10000);
367
368        D((0, "TNSClientWorkerThread\n"));
369
370        KeSetPriorityThread(KeGetCurrentThread(), LOW_REALTIME_PRIORITY+7);
371
372        //
373        //  Build and send out broadcast hello, and wait for a response.
374        //  We need to get the SMN mac address for future
375        //  transactions
376        //
377
378        //
379        //  Make sure driver has been initialized properly (this is
380        //  an assertion, this case should never happen)
381        //
382        //
383        //  Shack hack work on error handling
384        //
385        while (!pAdapter->TNSDriverInitialized) {
386            //
387            //  Wait until the driver has been completely initialized
388            //  then continue
389            //
390            KeDelayExecutionThread(
391                KernelMode,
392                FALSE,
393                &waittime);
394        }
395
396        //
397        //  Raise IRQL to prevent task swapping while we complete processing
398        //  for this packet.
399        //
400        //  KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);
401
402
403        if (TNSSharedMemoryNodeEmulation == FALSE) {
404            //
405            //  Compute packet length based on request, and
406            //  set the variable accordingly (the packet structure length
407            //  will get set according to this variable)
408            //
409
```

```
410          HelloRetryCount = 0;
411
412          while ( (HelloRetryCount++ < MAX_HELLO_RETRIES) && (HelloReceivedReply == FALSE) ) {
413
414               PacketLength = TNS_PACKET_SIZE(TNSPacketHelloBroadcast);
415               Status = TNSInitializeClientNodeSendPacket(pAdapter,
416                    &MyPacket,
417                    &pTnsBuffer,
418                    PacketLength);
419
420               D((0, "HelloRetryCount => %d\n", HelloRetryCount));
421               //
422               // Fill in relevent packet information here
423               //
424               pTnsBuffer->TNSCommandReply = wswap(TNS_HELLO_BROADCAST);
425
426               pTnsBuffer->RequestTag = dwswap(TNSGetRequestTag());
427               pTnsBuffer->RequestStartTSC = rdtsc();
428               for (i=0; i<6; i++) {
429                    pTnsBuffer->ClientMacAddress[i] = pAdapter->LowerMPMacAddress[i];
430               }
431               RtlCopyMemory(pTnsBuffer->ClientMachineName, pAdapter->LocalComputerName, MAX_COMPUTER_NAME_S
  -2 IZE);
432
433               if (NT_SUCCESS(Status)) {
434                    PLIST_ENTRY wrkrRequest;
435                    PREQUEST_DATA pWrkrRequestData;
436                    LARGE_INTEGER queueWait;
437
438                    //
439                    // Send request packet to SMN
440                    //
441                    TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
442
443                    //
444                    // This is a read operation, so we expect a response.
445                    // Block waiting for the response from the SMN.
446                    //
447                    queueWait.QuadPart = -(HelloRetryCount*1000*1000);
448
449                    Status = KeWaitForSingleObject(
450                         (PVOID) &pAdapter->ClientWorkerResponseSemaphore,
451                         Executive,
452                         KernelMode,
453                         FALSE,
454                         &queueWait);
455
456                    if (Status == STATUS_TIMEOUT) {
457                         //
458                         // Do something useful, like inc a stat
459                         //
460
461                    } else {
462                         //
463                         // We got a reply.
464                         //
465
466                         clientRequest = ExInterlockedRemoveHeadList(
467                              &serverContext->ClientWorkerListEntry,
468                              &serverContext->ClientWorkerListSpinLock);
469
470                         MyAssert(clientRequest != NULL);
471
472                         pClientRequestData = CONTAINING_RECORD(clientRequest,
473                                   REQUEST_DATA,
474                                   Linkage);
475
476                         MyAssert(pClientRequestData != NULL);
477
478                         if (pClientRequestData->requestOpcode != TNS_HELLO_REPLY) {
479                              MyAssert(0);
480                         } else {
481                              D((0, "We got a hello reply\n"));
482                              HelloReceivedReply = TRUE;
483                         }
484
485                         //
486                         // Recycle the queue object
487                         //
488                         ExInterlockedInsertTailList(&serverContext->WorkerListEntryPool,
489                              &pClientRequestData->Linkage,
490                              &serverContext->ListEntryPoolLock);
```

```
491
492                          )
493                       )
494              )
495
496        while (1) {
497             KeDelayExecutionThread(
498                  KernelMode,
499                  FALSE,
500                  &queueWait);
501
502             TnsGetNICStats(pAdapter, &pAdapter->mpStats);
503        }
504    }
505
506
507
508      PsTerminateSystemThread(STATUS_SUCCESS);
509 }
510
511
512
513 VOID
514 TNSServerWorkerThread(
515      PVOID Context
516      )
517 {
518      NTSTATUS waitStatus;
519      LARGE_INTEGER queueWait;
520      PADAPTER serverContext = (PADAPTER)Context;
521      PADAPTER pAdapter = (PADAPTER)Context;
522      PLIST_ENTRY serverRequest;
523      PREQUEST_DATA pServerRequestData;
524      NTSTATUS Status;
525
526      queueWait.QuadPart = -(3*1000*10000);
527
528      D((0, "TNSServerWorkerThread\n"));
529
530      if (TNSSharedMemoryNodeEmulation) {
531
532          pAdapter->TNSSharedMemoryPtr = NULL;
533          pAdapter->TNSSharedMemorySize = 0;
534
535          //ismitree
536          pAdapter->TNSMemoryType = VIRTUAL_MEMORY;
537          pAdapter->TNSMemoryType = NONPAGED_MEMORY;
538
539          if (pAdapter->TNSMemoryType == VIRTUAL_MEMORY) {
540              //
541              //Make 4 meg to start with
542              //
543
544              pAdapter->TNSSharedMemorySize = 1024*1024*4;
545
546              Status = ZwAllocateVirtualMemory(
547                  (HANDLE)    NtCurrentProcess(),
548                  (PVOID *)   &pAdapter->TNSSharedMemoryPtr,
549                  (ULONG)     0,
550                  (PULONG)    &pAdapter->TNSSharedMemorySize,
551                  (ULONG)     MEM_COMMIT,
552                  (ULONG)     PAGE_READWRITE);
553
554              if (Status != STATUS_SUCCESS) {
555                  D((0, "Virtual memory allocation failed\n"));
556                  _asm int 3
557              } else {
558                  D((0, "Virtual memory allocation succeeded\n"));
559                  RtlZeroMemory(pAdapter->TNSSharedMemoryPtr, pAdapter->TNSSharedMemorySize);
560              }
561          }
562          if (pAdapter->TNSMemoryType == NONPAGED_MEMORY) {
563              //
564              //Make 1 meg to start with
565              //
566              pAdapter->TNSSharedMemorySize = 1024*1024*1;
567
568              pAdapter->TNSSharedMemoryPtr =
569                  ExAllocatePool(
570                      NonPagedPool,
571                      pAdapter->TNSSharedMemorySize);
572
```

```
573                 if (pAdapter->TNSSharedMemoryPtr == NULL) {
574                     D((0, "NonPagedPool memory allocation failed\n"));
575                     _asm int 3
576                 } else {
577                     D((0, "NonPagedPool  memory allocation succeeded\n"));
578                     RtlZeroMemory(pAdapter->TNSSharedMemoryPtr, pAdapter->TNSSharedMemorySize);
579                 }
580
581             }
582
583         }
584         KeSetPriorityThread(KeGetCurrentThread(), LOW_REALTIME_PRIORITY+7);
585
586         do {
587             waitStatus = KeWaitForSingleObject(
588                 (PVOID) &serverContext->ServerWorkerRequestSemaphore,
589                 Executive,
590                 KernelMode,
591                 FALSE,
592                 &queueWait);
593
594
595             //
596             // Check for timeout, if we do, then do something
597             //
598             if (waitStatus == STATUS_TIMEOUT) {
599                 //
600                 // If status is timeout, take the opportunity to do something useful,
601                 // and collect some statistical data
602                 //
603                 TnsGetNICStats(pAdapter, &pAdapter->mpStats);
604
605                 continue;
606             }
607
608             //D((0, "TNSServerWorkerThread - dequeued an object\n"));
609             MyAssert(waitStatus == STATUS_SUCCESS);
610
611             while (serverRequest = ExInterlockedRemoveHeadList(
612                 &serverContext->ServerWorkerListEntry,
613                 &serverContext->ServerWorkerListSpinLock)) {
614
615                 pServerRequestData = CONTAINING_RECORD(serverRequest,
616                             REQUEST_DATA,
617                             Linkage);
618
619                 MyAssert(pServerRequestData);
620
621                 switch (pServerRequestData->requestOpcode) {
622                     case TNS_READ_REQUEST: {
623                         PNDIS_PACKET MyPacket;
624                         ULONG PacketLength;
625                         PTNSPacketReadReply pTnsBuffer;
626                         NTSTATUS Status;
627                         PUCHAR   vBuffer;
628
629                         vBuffer = pAdapter->TNSSharedMemoryPtr;
630
631                         //D((0, "Processing server read request\n"));
632                         PacketLength = TNS_PACKET_SIZE(TNSPacketReadReply);
633
634                         Status = TNSInitializeClientNodeSendPacket(pAdapter,
635                             &MyPacket,
636                             &pTnsBuffer,
637                             PacketLength);
638
639                         RtlCopyMemory(pTnsBuffer, &((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->
-2 MACSrcAddress, 6);
640                         //
641                         // Fill in relevant packet information here
642                         //
643                         pTnsBuffer->TNSCommandReply = wswap(TNS_READ_REPLY);
644
645                         pTnsBuffer->RequestTag = ((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->Re
-2 questTag;
646                         pTnsBuffer->RequestStartTSC = ((PTNSPacketReadRequest)(pServerRequestData->TnsPacket)
-2 )->RequestStartTSC;
647                         vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap(((PTNSPacketReadRequest)(pServerReque
-2 stData->TnsPacket))->RequestOffset));
648
649                         if (dwswap( ((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->RequestOffset)
-2 <= pAdapter->TNSSharedMemorySize ) {
```

```
650                        pTnsBuffer->dwData = *((PULONG)vBuffer);
651                    } else {
652                        _asm int 3
653                    }
654
655                    TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
656
657                    break;
658                }
659            case TNS_WRITE_REQUEST: {
660                PNDIS_PACKET MyPacket;
661                ULONG PacketLength;
662                NTSTATUS Status;
663                PUCHAR   vBuffer;
664
665                //D((0, "Processing server write request\n"));
666
667                vBuffer = pAdapter->TNSSharedMemoryPtr;
668
669                vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap( ((PTNSPacketWriteRequest)(pServerReq
-2 uestData->TnsPacket))->RequestOffset)));
670
671                if (dwswap(((PTNSPacketWriteRequest)(pServerRequestData->TnsPacket))->RequestOffset)
-2 <= pAdapter->TNSSharedMemorySize ) {
672                    *((PULONG)vBuffer) = ((PTNSPacketWriteRequest)(pServerRequestData->TnsPacket))->d
-2 wData;
673                } else {
674                    _asm int 3
675                }
676
677                break;
678            }
679            case TNS_HELLO_REPLY:
680                MyAssert(TNSSharedMemoryNodeEmulation);
681                //
682                // Send hello reply
683                //
684                D((0, "Processing server hello reply\n"));
685
686                TNSSendPackets(pAdapter->LowerMPHandle, &pServerRequestData->pNdisPacket, 1);
687
688                break;
689            default:
690                MyAssert(0);
691                break;
692            }
693            //
694            // Recycle the queue object
695            //
696            ExInterlockedInsertTailList(&serverContext->WorkerListEntryPool,
697                &pServerRequestData->Linkage,
698                &serverContext->ListEntryPoolLock);
699        }
700    } while (TRUE);
701
702    PsTerminateSystemThread(STATUS_SUCCESS);
703 }
704
705 VOID
706 TNSEmulSetPacketHeader(
707     PADAPTER    pAdapter,
708     PVOID       pTnsPacket,
709     UINT        PacketLength)
710 {
711     UINT i;
712     ULONG *pulData;
713
714     pulData = (PULONG) pTnsPacket;
715
716     //
717     // Zero memory (take this out later)
718     //
719     RtlZeroMemory(pTnsPacket, PacketLength);
720
721     //
722     // Put a recognizable pattern into packet buffer
723     //
724     for (i=0; i<PacketLength/4; i++) {
725         *pulData++ = 0xcafebabe;
726     }
727
728     //
```

```
729      // Set the destination and source addresses for the packet
730      //
731      for (i=0; i<HARDWARE_ADDRESS_LENGTH; i++) {
732          ((PTNSPacketHeader)pTnsPacket)->MACDstAddress[i] = pAdapter->SMNMacAddress[i];
733          ((PTNSPacketHeader)pTnsPacket)->MACSrcAddress[i] = pAdapter->LowerMPMacAddress[i];
734      }
735      //
736      // Set the ethertype to our ethertype
737      //
738      ((PTNSPacketHeader)pTnsPacket)->MACEtherType = wswap(TNS_EMULATION_ETHERTYPE);
739
740  }
741
742  //
743  // Initialized to 0, incremented by 1 each time we use it.  We use
744  // this to help up keep track of outstanding requests to the SMN.
745  //
746  unsigned long _gRequestTag = 0;
747  unsigned long
748  TNSGetRequestTag(void)
749  {
750      return _gRequestTag++;
751  }
752
753
754  //
755  // Initialized to 0, incremented by 1 each time we use it.  We use
756  // this to help up keep track of outstanding requests to the SMN.
757  //
758  unsigned long _gSharedMemoryNodeNodeID = 0;
759  unsigned long
760  TNSGetSharedMemoryNodeNodeID(
761      PADAPTER pAdapter,
762      unsigned char *pHeader)
763  {
764      ULONG i;
765      ULONG NextFreeSpace=0xffffffff;
766      ULONG NewTeamNodeID;
767      PTNSPacketHelloBroadcast pTnsPacket = (PTNSPacketHelloBroadcast) pHeader;
768
769      for (i=0; i<MAX_TEAM_NODES; i++) {
770          if (pAdapter->TeamNodeTable[i].LocationSet) {
771              if ( RtlCompareMemory(&pHeader[6], pAdapter->TeamNodeTable[i].TNMacAddress, 6) == 6) {
772                  return pAdapter->TeamNodeTable[i].TNNodeID;
773              }
774          } else {
775              if (NextFreeSpace == 0xffffffff) {
776                  NextFreeSpace = i;
777              }
778          }
779      }
780
781      //
782      // If we made it this far, we did not find an entry.
783      // Set an entry in our table for this mac address.
784      //
785      NewTeamNodeID = _gSharedMemoryNodeNodeID++;
786      RtlCopyMemory(pAdapter->TeamNodeTable[NextFreeSpace].TNMacAddress, &pHeader[6], 6);
787      RtlCopyMemory(pAdapter->TeamNodeTable[NextFreeSpace].TNComputerName, pTnsPacket->ClientMachineName, M
-2 AX_COMPUTER_NAME_SIZE);
788      pAdapter->TeamNodeTable[NextFreeSpace].LocationSet = TRUE;
789      pAdapter->TeamNodeTable[NextFreeSpace].TNNodeID = NewTeamNodeID;
790
791      return NewTeamNodeID;
792  }
793
794
795  LARGE_INTEGER diffTime;
796
797  NTSTATUS
798  TNSInitializeClientNodeSendPacket(
799      IN      PADAPTER      pAdapter,
800      IN OUT  PNDIS_PACKET  *ppNdisPacket,
801      IN OUT  PVOID         *ppTnsBuffer,
802      IN      ULONG         PacketLength)
803  {
804      NTSTATUS Status=STATUS_SUCCESS;
805      PTNS_PACKET_CONTEXT    PktContext;
806      PNDIS_PACKET    MyPacket;
807      PNDIS_BUFFER    MyNdisBuffer;
808      PVOID       vBuffer;
809      NDIS_PHYSICAL_ADDRESS HighAddress = NDIS_PHYSICAL_ADDRESS_CONST( -1, -1 );
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c**                       **Page 11 of 39**

```
810     PVOID pTnsPacket;
811     LARGE_INTEGER startTime, endTime;
812
813     //
814     // Allocate a packet from our available packet pool.
815     // Check status, reinit the packet, and get the
816     // context context area
817     //
818     startTime = rdtsc();
819     NdisAllocatePacket(&Status, &MyPacket, pAdapter->PacketPoolHandle);
820     endTime = rdtsc();
821
822     diffTime.QuadPart = endTime.QuadPart - startTime.QuadPart;
823
824     if (diffTime.LowPart > 0x400) {
825         //DbgOut("NdisAllocatePacket Time => %x\n", diffTime.LowPart));
826     }
827
828     //
829     // hack hack work on error handling
830     //
831     if (Status != STATUS_SUCCESS) {
832         _asm int 3
833         return Status;
834     }
835     NdisReinitializePacket(MyPacket);
836
837     PktContext = PACKET_CONTEXT_FROM_PACKET(MyPacket);
838
839     PktContext->OriginalPacket = NULL;
840     PktContext->LookaheadBuffer = NULL;
841     PktContext->SMNEmulationPacket = TRUE;
842
843     //
844     // Now, allocate a buffer to chain to the packet
845     //
846     Status = NdisAllocateMemory(&vBuffer, PacketLength, 0, HighAddress);
847
848     //
849     // hack hack work on error handling
850     //
851     if (Status != NDIS_STATUS_SUCCESS) {
852         NdisFreePacket(MyPacket);
853         return Status;
854     }
855
856     NdisAllocateBuffer(&Status,
857         &MyNdisBuffer,
858         pAdapter->LookaheadPoolHandle,
859         vBuffer,
860         PacketLength);
861
862     //
863     // hack hack work on error handling
864     //
865     if (Status != NDIS_STATUS_SUCCESS) {
866         _asm int 3
867         NdisFreePacket(MyPacket);
868         NdisFreeMemory(vBuffer, PacketLength, 0);
869         return Status;
870     }
871
872     pTnsPacket = (PTNSPacketHelloBroadcast) vBuffer;
873
874     //
875     // Setup the packet mac dest, source, and ethertype
876     //
877
878     TNSEmulSetPacketHeader(pAdapter, pTnsPacket, PacketLength);
879
880     //
881     // Set the packet length
882     //
883     NdisAdjustBufferLength(MyNdisBuffer, PacketLength);
884
885     //
886     // Chain our buffer to this packet structure
887     //
888     NdisChainBufferAtFront(MyPacket, MyNdisBuffer);
889     NdisRecalculatePacketCounts(MyPacket);
890
891     *ppNdisPacket = MyPacket;
```

```
892      *ppTnsBuffer = pTnsPacket;
893
894      return Status;
895 }
896
897 VOID
898 TNSFlushReadReplyQueue(
899     PADAPTER pAdapter)
900 {
901     LARGE_INTEGER queueWait;
902     NTSTATUS Status;
903     PLIST_ENTRY clientRequest;
904     PREQUEST_DATA pClientRequestData;
905
906     do {
907         queueWait.QuadPart = -(0);
908
909         Status = KeWaitForSingleObject(
910             (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
911             Executive,
912             KernelMode,
913             FALSE,
914             &queueWait);
915
916         if (Status == STATUS_SUCCESS) {
917
918             clientRequest = ExInterlockedRemoveHeadList(
919                 &pAdapter->ClientWorkerListEntry,
920                 &pAdapter->ClientWorkerListSpinLock);
921
922             MyAssert(clientRequest != NULL);
923
924             pClientRequestData = CONTAINING_RECORD(clientRequest,
925                         REQUEST_DATA,
926                         Linkage);
927
928             MyAssert(pClientRequestData);
929
930             TnsIncrementStat(pAdapter, &pAdapter->MyStats.numDiscardedTnsRecvs);
931
932             //
933             // Recycle the queue object
934             //
935             ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
936                 &pClientRequestData->Linkage,
937                 &pAdapter->ListEntryPoolLock);
938         }
939     } while (Status == STATUS_SUCCESS) ;
940 }
941
942
943
944 //
945 // Start Kernel-Mode DLL entry points.
946 //
947
948 #define MAX_REQUEST_RESPONSE_RETRIES    50
949
950 //
951 //
952 ULONG
953 DECLSPEC_EXPORT
954 __TNS_READ_REGISTER_ULONG(
955     IN  PVOID   DeviceHandle,
956     IN  PULONG  Register)
957 //
958 // Description:
959 //
960 // Environment:
961 //
962 // Return Value:
963 //
964 //
965 //
966 {
967     ULONG   RegisterData=0xbaddc0de;
968     PADAPTER pAdapter = (PADAPTER) DeviceHandle;
969     NTSTATUS    Status;
970     KIRQL   OldIrql;
971     PNDIS_PACKET MyPacket;
972     ULONG PacketLength;
973     PTNSPacketReadRequest pTnsBuffer;
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c          **Page 13 of 3**

```
974      PLIST_ENTRY clientRequest;
975      PREQUEST_DATA pClientRequestData;
976      ULONG requestTag;
977      ULONG retries=0;
978      int noreply = TRUE;
979      ULONG returnRequestTag;
980      LARGE_INTEGER startTime, endTime, diffTime;
981
982      //
983      //hack hack. We really wanna use the device context given up
984      //by the caller.
985      //
986      pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
987
988      //
989      //Raise IRQL to prevent task swapping while we complete processing
990      //for this packet.
991      //
992      KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);
993
994      //
995      //Make sure driver has been intialized properly (this is
996      //an assertion, this case should never happen).
997      //
998      //
999      //hack hack work on error handling
1000     //
1001     if (!pAdapter->TNSDriverInitialized) {
1002         BreakPoint();
1003         KeLowerIrql(OldIrql);
1004         return 0;
1005     }
1006
1007     TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadRequests);
1008     //
1009     //compute packet length, based on request, and
1010     //set the variable accordingly (the packet structure length
1011     //will get set according to this variable).
1012     //
1013
1014     PacketLength = TNS_PACKET_SIZE(TNSPacketReadRequest);
1015
1016     requestTag = TNSGetRequestTag();
1017
1018     while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {
1019
1020         Status = TNSInitializeClientNodeSendPacket(pAdapter,
1021             &MyPacket,
1022             &pTnsBuffer,
1023             PacketLength);
1024
1025         //
1026         //Fill in relevant packet information here.
1027         //
1028         pTnsBuffer->TNSCommandReply = wswap(TNS_READ_REQUEST);
1029
1030         pTnsBuffer->RequestTag = dwswap(requestTag);
1031         pTnsBuffer->RequestWidth = dwswap(4);
1032         pTnsBuffer->RequestLength = dwswap(1);
1033         pTnsBuffer->RequestOffset = dwswap((unsigned long)Register);
1034         pTnsBuffer->RequestStartTSC = rdtsc();
1035
1036         if (NT_SUCCESS(Status)) {
1037             PLIST_ENTRY wrkrRequest;
1038             PREQUEST_DATA pWrkrRequestData;
1039             LARGE_INTEGER queueWait;
1040             int timeout = FALSE;
1041             int ltimeout = FALSE;
1042             int timeoutcount = 0;
1043
1044             //
1045             //Flush the read reply queue. In case a different request timed out,
1046             //and it actually shows up, we need to flush the queue for
1047             //subsequent requests.
1048             //
1049             TNSFlushReadReplyQueue(pAdapter);
1050
1051             startTime = rdtsc();
1052             //
1053             //Send request packet to SMN
1054             //
1055             TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
```

```
1056
1057          //
1058          // This is a read operation, so we expect a response.
1059          // Block waiting for the response from the SMN.
1060          //
1061          // this is 100m secs.
1062          //
1063
1064          queueWait.QuadPart = -(1000000);
1065
1066          Status = KeWaitForSingleObject(
1067              (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
1068              Executive,
1069              KernelMode,
1070              FALSE,
1071              &queueWait);
1072
1073          if (Status != STATUS_TIMEOUT) {
1074              PTNSPacketReadReply pTnsPacketReadReply;
1075
1076              clientRequest = ExInterlockedRemoveHeadList(
1077                  &pAdapter->ClientWorkerListEntry,
1078                  &pAdapter->ClientWorkerListSpinLock);
1079
1080              MyAssert(clientRequest != NULL);
1081
1082              pClientRequestData = CONTAINING_RECORD(clientRequest,
1083                          REQUEST_DATA,
1084                          Linkage);
1085
1086              MyAssert(pClientRequestData != NULL);
1087              pTnsPacketReadReply = (PTNSPacketReadReply) &pClientRequestData->TnsPacket;
1088
1089              RegisterData     = pTnsPacketReadReply->dwData;
1090              returnRequestTag = dwswap(pTnsPacketReadReply->RequestTag);
1091
1092              //MyAssert(returnRequestTag == requestTag);
1093
1094              if (returnRequestTag == requestTag) {
1095                  noreply = FALSE;
1096                  endTime = rdtsc();
1097              }
1098
1099              //
1100              // only maintain stats if we did not retry the operation
1101              //
1102              if ( (retries == 1) && (noreply==FALSE) ) {
1103                  diffTime.QuadPart = endTime.QuadPart - startTime.QuadPart;
1104                  if (pAdapter->MyStats.maxReadTimeSingle.QuadPart == 0) {
1105                      pAdapter->MyStats.maxReadTimeSingle.QuadPart = diffTime.QuadPart;
1106                  } else {
1107                      if (diffTime.QuadPart > pAdapter->MyStats.maxReadTimeSingle.QuadPart) {
1108                          pAdapter->MyStats.maxReadTimeSingle.QuadPart = diffTime.QuadPart;
1109                      }
1110                  }
1111                  if (pAdapter->MyStats.minReadTimeSingle.QuadPart == 0) {
1112                      pAdapter->MyStats.minReadTimeSingle.QuadPart = diffTime.QuadPart;
1113                  } else {
1114                      if (diffTime.QuadPart < pAdapter->MyStats.minReadTimeSingle.QuadPart) {
1115                          pAdapter->MyStats.minReadTimeSingle.QuadPart = diffTime.QuadPart;
1116                      }
1117                  }
1118                  if (pAdapter->MyStats.numReadTimeSingleSamples.QuadPart < 30000) {
1119                      pAdapter->MyStats.cumReadTimeSingle.QuadPart += diffTime.QuadPart;
1120                      TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadTimeSingleSamples);
1121                  } else {
1122                      pAdapter->MyStats.cumReadTimeSingle.QuadPart = diffTime.QuadPart;
1123                      pAdapter->MyStats.numReadTimeSingleSamples.QuadPart = 1;
1124                  }
1125              }
1126
1127              //
1128              // Recycle the queue object
1129              //
1130              ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
1131                  &pClientRequestData->Linkage,
1132                  &pAdapter->ListEntryPoolLock);
1133          } else {
1134              TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadRequestTimeouts);
1135          }
1136      }
1137  }
```

```
1138
1139        KeLowerIrql(OldIrql);
1140
1141        if (retries > 1) {
1142            TnsAddStatsUlong(pAdapter, &pAdapter->MyStats.numWriteRequestRetries, retries-1);
1143        }
1144
1145        if (noreply == TRUE) {
1146            RegisterData = 0xFFFFFFFF;
1147
1148            TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadRequestNoReplies);
1149            //
1150            // Throw an exception to our client
1151            //
1152            // TODO
1153        }
1154
1155        return RegisterData;
1156 }
1157
1158
1159
1160 //*****************************************************************************
1161 //*+
1162 VOID
1163 DECLSPEC_EXPORT
1164 __TNS_WRITE_REGISTER_ULONG(
1165        IN   PVOID    DeviceHandle,
1166        IN   PULONG   Register,
1167        IN   ULONG    RegisterData)
1168 //
1169 // Description:
1170 //
1171 // Environment:
1172 //
1173 // Return Value:
1174 //
1175 //*-
1176 //*****************************************************************************
1177 {
1178        PADAPTER pAdapter = (PADAPTER) DeviceHandle;
1179        NTSTATUS    Status;
1180        KIRQL   OldIrql;
1181        PNDIS_PACKET MyPacket;
1182        ULONG PacketLength;
1183        PTNSPacketWriteRequest pTnsBuffer;
1184        ULONG requestTag;
1185        ULONG retries=0;
1186        int noreply = TRUE;
1187        PLIST_ENTRY clientRequest;
1188        PREQUEST_DATA pClientRequestData;
1189        ULONG returnRequestTag;
1190        LARGE_INTEGER startTime, endTime, diffTime;
1191
1192
1193        // DT(0, "TNS_WRITE_REGISTER_ULONG\n");
1194
1195        //
1196        // hack hack  we really wanna use the device context given up
1197        // by the caller.
1198        //
1199        pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
1200
1201        //
1202        // Raise IRQL to prevent task swapping while we complete processing
1203        // for this packet.
1204        //
1205        KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);
1206
1207        //
1208        // Make sure driver has been initialized properly (this is
1209        // an assertion, this case should never happen).
1210        //
1211        //
1212        // hack hack work on error handling
1213        //
1214        if (!pAdapter->TNSDriverInitialized) {
1215            BreakPoint();
1216            KeLowerIrql(OldIrql);
1217            return;
1218        }
1219
```

```
1220        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteRequests);
1221
1222        //
1223        // compute packet length (based on request) and
1224        // set the variable accordingly (the packet structure length
1225        // will get set according to this variable).
1226        //
1227
1228        PacketLength = TNS_PACKET_SIZE(TNSPacketWriteRequest);
1229        requestTag = TNSGetRequestTag();
1230
1231        while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {
1232
1233            Status = TNSInitializeClientNodeSendPacket(pAdapter,
1234                &MyPacket,
1235                &pTnsBuffer,
1236                PacketLength);
1237
1238            //
1239            // Fill in relevent packet information here....
1240            //
1241            pTnsBuffer->TNSCommandReply = wswap(TNS_WRITE_REQUEST);
1242
1243            pTnsBuffer->RequestTag = dwswap(requestTag);
1244            pTnsBuffer->RequestWidth = dwswap(4);
1245            pTnsBuffer->RequestLength = dwswap(1);
1246            pTnsBuffer->RequestOffset = dwswap((unsigned long)Register);
1247            pTnsBuffer->dwData = RegisterData;
1248            pTnsBuffer->RequestStartTSC = rdtsc();
1249
1250            if (NT_SUCCESS(Status)) {
1251                PLIST_ENTRY wrkrRequest;
1252                PREQUEST_DATA pWrkrRequestData;
1253                LARGE_INTEGER queueWait;
1254
1255                TNSFlushReadReplyQueue(pAdapter);
1256
1257                startTime = rdtsc();
1258                //
1259                // Send request packet to SMN (We <ass>ume realiable delivery)
1260                //
1261                TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
1262
1263                queueWait.QuadPart = -(1000000);
1264
1265                Status = KeWaitForSingleObject(
1266                    (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
1267                    Executive,
1268                    KernelMode,
1269                    FALSE,
1270                    &queueWait);
1271
1272                if (Status != STATUS_TIMEOUT) {
1273                    PTNSPacketWriteReply pTnsWriteReplyPacket;
1274
1275                    clientRequest = ExInterlockedRemoveHeadList(
1276                        &pAdapter->ClientWorkerListEntry,
1277                        &pAdapter->ClientWorkerListSpinLock);
1278
1279                    MyAssert(clientRequest != NULL);
1280
1281                    pClientRequestData = CONTAINING_RECORD(clientRequest,
1282                                REQUEST_DATA,
1283                                Linkage);
1284
1285                    MyAssert(pClientRequestData != NULL);
1286
1287                    pTnsWriteReplyPacket = (PTNSPacketWriteReply)&pClientRequestData->TnsPacket;
1288
1289                    returnRequestTag = dwswap(pTnsWriteReplyPacket->RequestTag);
1290
1291                    //MyAssert(returnRequestTag == requestTag);
1292
1293                    if (returnRequestTag == requestTag) {
1294                        noreply = FALSE;
1295                        endTime = rdtsc();
1296                    }
1297
1298                    if ( (retries == 1) && (noreply==FALSE) ) {
1299                        diffTime.QuadPart = endTime.QuadPart - startTime.QuadPart;
1300                        if (pAdapter->MyStats.maxWriteTimeSingle.QuadPart == 0) {
1301                            pAdapter->MyStats.maxWriteTimeSingle.QuadPart = diffTime.QuadPart;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c**　　　　　　　　　　　**Page 17 of 39**

```
1302                    } else {
1303                        if (diffTime.QuadPart > pAdapter->MyStats.maxWriteTimeSingle.QuadPart) {
1304                            pAdapter->MyStats.maxWriteTimeSingle.QuadPart = diffTime.QuadPart;
1305                        }
1306                    }
1307                    if (pAdapter->MyStats.minWriteTimeSingle.QuadPart == 0) {
1308                        pAdapter->MyStats.minWriteTimeSingle.QuadPart = diffTime.QuadPart;
1309                    } else {
1310                        if (diffTime.QuadPart < pAdapter->MyStats.minWriteTimeSingle.QuadPart) {
1311                            pAdapter->MyStats.minWriteTimeSingle.QuadPart = diffTime.QuadPart;
1312                        }
1313                    }
1314
1315                    if (pAdapter->MyStats.numWriteTimeSingleSamples.QuadPart < 30000) {
1316                        pAdapter->MyStats.cumWriteTimeSingle.QuadPart += diffTime.QuadPart;
1317                        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteTimeSingleSamples);
1318                    } else {
1319                        pAdapter->MyStats.cumWriteTimeSingle.QuadPart = diffTime.QuadPart;
1320                        pAdapter->MyStats.numWriteTimeSingleSamples.QuadPart = 1;
1321                    }
1322                }
1323
1324                //
1325                // Recycle the queue object
1326                //
1327                ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
1328                    &pClientRequestData->Linkage,
1329                    &pAdapter->ListEntryPoolLock);
1330
1331            } else {
1332
1333                TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteRequestTimeouts);
1334
1335            }
1336        }
1337    }
1338
1339    if (retries > 1) {
1340        TnsAddStatsUlong(pAdapter, &pAdapter->MyStats.numWriteRequestRetries, retries-1);
1341    }
1342
1343
1344    if (noreply == TRUE) {
1345        //
1346        // Throw an exception to our client   TODO
1347        //
1348
1349
1350        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteRequestNoReplies);
1351    }
1352
1353    KeLowerIrql(OldIrql);
1354
1355    return;
1356 }
1357
1358 //*******************************************************************
1359 //-+
1360 USHORT
1361 DECLSPEC_EXPORT
1362 __TNS_READ_REGISTER_USHORT(
1363     IN  PVOID   DeviceHandle,
1364     IN  PUSHORT Register)
1365 //
1366 // Description:
1367 //
1368 // Environment:
1369 //
1370 // Return Value:
1371 //
1372 //
1373 //*******************************************************************
1374 {
1375     USHORT RegisterData=0xbadd;
1376
1377     return RegisterData;
1378 }
1379
1380 //*******************************************************************
1381 //-+
1382 VOID
1383 DECLSPEC_EXPORT
```

```
1384 __TNS_WRITE_REGISTER_USHORT(
1385       IN  PVOID   DeviceHandle,
1386       IN  PUSHORT Register,
1387       IN  USHORT  RegisterData)
1388 //
1389 // Description:
1390 //
1391 // Environment:
1392 //
1393 // Return Value:
1394 //
1395 //
1396 //*****************************************************
1397 {
1398 }
1399
1400 //*****************************************************
1401 //
1402 UCHAR
1403 DECLSPEC_EXPORT
1404 __TNS_READ_REGISTER_UCHAR(
1405       IN  PVOID   DeviceHandle,
1406       IN  PUCHAR  Register)
1407 //
1408 // Description:
1409 //
1410 // Environment:
1411 //
1412 // Return Value:
1413 //
1414 //
1415 //*****************************************************
1416 {
1417       UCHAR  RegisterData=0xba;
1418
1419       return RegisterData;
1420 }
1421
1422 //*****************************************************
1423 //
1424 VOID
1425 DECLSPEC_EXPORT
1426 __TNS_WRITE_REGISTER_UCHAR(
1427       IN  PVOID   DeviceHandle,
1428       IN  PUCHAR  Register,
1429       IN  UCHAR   RegisterData)
1430 //
1431 // Description:
1432 //
1433 // Environment:
1434 //
1435 // Return Value:
1436 //
1437 //
1438 //*****************************************************
1439 {
1440 }
1441
1442
1443
1444 //*****************************************************
1445 //
1446 VOID
1447 DECLSPEC_EXPORT
1448 __TNS_READ_REGISTER_BUFFER_ULONG(
1449       IN  PVOID   DeviceHandle,
1450       IN  PULONG  Register,
1451       IN  PULONG  pulBuffer,
1452       IN  ULONG   Count)
1453 //
1454 // Description:
1455 //
1456 // Environment:
1457 //
1458 // Return Value:
1459 //
1460 //
1461 //*****************************************************
1462 {
1463 }
1464
1465 //*****************************************************
```

```
1466 //:*
1467 VOID
1468 DECLSPEC_EXPORT
1469 __TNS_WRITE_REGISTER_BUFFER_ULONG(
1470      IN  PVOID   DeviceHandle,
1471      IN  PULONG  Register,
1472      IN  PULONG  pulBuffer,
1473      IN  ULONG   Count)
1474 //
1475 // Description:
1476 //
1477 // Environment:
1478 //
1479 // Return Value:
1480 //
1481 //
1482 //*****************************************************************
1483 {
1484 }
1485
1486 //*****************************************************************
1487 //:*
1488 VOID
1489 DECLSPEC_EXPORT
1490 __TNS_READ_REGISTER_BUFFER_USHORT(
1491      IN  PVOID   DeviceHandle,
1492      IN  PUSHORT Register,
1493      IN  PUSHORT pusBuffer,
1494      IN  ULONG   Count)
1495 //
1496 // Description:
1497 //
1498 // Environment:
1499 //
1500 // Return Value:
1501 //
1502 //
1503 //*****************************************************************
1504 {
1505 }
1506
1507 //*****************************************************************
1508 //:*
1509 VOID
1510 DECLSPEC_EXPORT
1511 __TNS_WRITE_REGISTER_BUFFER_USHORT(
1512      IN  PVOID   DeviceHandle,
1513      IN  PUSHORT Register,
1514      IN  PUSHORT pusBuffer,
1515      IN  ULONG   Count)
1516 //
1517 // Description:
1518 //
1519 // Environment:
1520 //
1521 // Return Value:
1522 //
1523 //
1524 //*****************************************************************
1525 {
1526 }
1527
1528
1529
1530
1531 //*****************************************************************
1532 //:*
1533 VOID
1534 DECLSPEC_EXPORT
1535 __TNS_READ_REGISTER_BUFFER_UCHAR(
1536      IN  PVOID   DeviceHandle,
1537      IN  PUCHAR  Register,
1538      IN  PUCHAR  pucBuffer,
1539      IN  ULONG   Count)
1540 //
1541 // Description:
1542 //
1543 // Environment:
1544 //
1545 // Return Value:
1546 //
1547 //
```

```
1548 //*************************************************************************
1549 {
1550 }
1551
1552 //*************************************************************************
1553 //--
1554 VOID
1555 DECLSPEC_EXPORT
1556 __TNS_WRITE_REGISTER_BUFFER_UCHAR(
1557      IN  PVOID   DeviceHandle,
1558      IN  PUCHAR  Register,
1559      IN  PUCHAR  pucBuffer,
1560      IN  ULONG   Count)
1561 //
1562 // Description:
1563 //
1564 // Environment:
1565 //
1566 // Return Value:
1567 //
1568 //--
1569 //*************************************************************************
1570 {
1571 }
1572
1573 //*************************************************************************
1574 //--
1575 TNS_STATUS
1576 DECLSPEC_EXPORT
1577 __TNSAcquireLockP(
1578      IN  PVOID   DeviceHandle,
1579      IN  PLOCKID pLockID)
1580 //
1581 // Description:
1582 //
1583 // Environment:
1584 //
1585 // Return Value:
1586 //
1587 //--
1588 //*************************************************************************
1589 {
1590     return TNS_STATUS_NOT_IMPLEMENTED;
1591 }
1592
1593 //*************************************************************************
1594 //--
1595 TNS_STATUS
1596 DECLSPEC_EXPORT
1597 __TNSReleaseLockP(
1598      IN  PVOID   DeviceHandle,
1599      IN  PLOCKID pLockID)
1600 //
1601 // Description:
1602 //
1603 // Environment:
1604 //
1605 // Return Value:
1606 //
1607 //--
1608 //*************************************************************************
1609 {
1610     return TNS_STATUS_NOT_IMPLEMENTED;
1611 }
1612
1613 //*************************************************************************
1614 //--
1615 TNS_STATUS
1616 DECLSPEC_EXPORT
1617 __TNSQueryLockP(
1618      IN  PVOID       DeviceHandle,
1619      OUT PLOCKSTATUS pLockStatus)
1620 //
1621 // Description:
1622 //
1623 // Environment:
1624 //
1625 // Return Value:
1626 //
1627 //--
1628 //*************************************************************************
1629 {
```

```
1630       return TNS_STATUS_NOT_IMPLEMENTED;
1631 }
1632
1633
1634
1635 //********************************************************************
1636 //
1637 TNS_STATUS
1638 DECLSPEC_EXPORT
1639 __TNSAllocateLockP(
1640     IN  PVOID   DeviceHandle,
1641     IN  TNSKEY  Key,
1642     OUT PLOCKID *pLockID)
1643 //
1644 // Description:
1645 //
1646 // Environment:
1647 //
1648 // Return Value:
1649 //
1650 //
1651 //********************************************************************
1652 {
1653       return TNS_STATUS_NOT_IMPLEMENTED;
1654 }
1655
1656 //********************************************************************
1657 //
1658 TNS_STATUS
1659 DECLSPEC_EXPORT
1660 __TNSFreeLockP(
1661     IN  PVOID   DeviceHandle,
1662     IN  TNSKEY  Key,
1663     IN  PLOCKID pLockID)
1664 //
1665 // Description:
1666 //
1667 // Environment:
1668 //
1669 // Return Value:
1670 //
1671 //
1672 //********************************************************************
1673 {
1674       return TNS_STATUS_NOT_IMPLEMENTED;
1675 }
1676
1677 //********************************************************************
1678 //
1679 TNS_STATUS
1680 DECLSPEC_EXPORT
1681 __TNSNotifyCPU(
1682     IN  PVOID       DeviceHandle,
1683     IN  TNSCPUID    CpuID,
1684     IN  PVOID       pMessageBuffer,
1685     IN  ULONG       MessageLength)
1686 //
1687 // Description:
1688 //
1689 // Environment:
1690 //
1691 // Return Value:
1692 //
1693 //
1694 //********************************************************************
1695 {
1696       return TNS_STATUS_NOT_IMPLEMENTED;
1697 }
1698
1699 //********************************************************************
1700 //
1701 TNS_STATUS
1702 DECLSPEC_EXPORT
1703 __TNSNotifyCPUSync(
1704     IN  PVOID       DeviceHandle,
1705     IN  TNSCPUID    CpuID,
1706     IN  PVOID       pMessageBuffer,
1707     IN  ULONG       MessageLength,
1708     IN  PVOID       pCallback,
1709     IN  PVOID       pContext)
1710 //
1711 // Description:
```

```
1712  //
1713  // Environment:
1714  //
1715  // Return Value:
1716  //
1717  //--
1718  //*************************************************************
1719  {
1720       return TNS_STATUS_NOT_IMPLEMENTED;
1721  }
1722
1723
1724  //*************************************************************
1725  //--
1726  TNS_STATUS
1727  DECLSPEC_EXPORT
1728  __TNSQueryNotifyStatus(
1729       IN      PVOID              DeviceHandle,
1730       IN      TNSCPUID           CpuID,
1731       IN  OUT PTNSNOTIFYSTATUS   pCpuNotifyInfo)
1732  //
1733  // Description:
1734  //
1735  // Environment:
1736  //
1737  // Return Value:
1738  //
1739  //--
1740  //*************************************************************
1741  {
1742       return TNS_STATUS_NOT_IMPLEMENTED;
1743  }
1744
1745
1746  //*************************************************************
1747  //--
1748  TNS_STATUS
1749  DECLSPEC_EXPORT
1750  __TNSRegisterNotifyCallback(
1751       IN  PVOID      DeviceHandle,
1752       IN  PVOID      pCallBack,
1753       IN  PVOID      SysParm1,
1754       IN  PVOID      SysParm2,
1755       IN  PVOID      SysParm3)
1756  //
1757  // Description:
1758  //
1759  // Environment:
1760  //
1761  // Return Value:
1762  //
1763  //--
1764  //*************************************************************
1765  {
1766       return TNS_STATUS_NOT_IMPLEMENTED;
1767  }
1768
1769
1770  //*************************************************************
1771  //--
1772  TNS_STATUS
1773  DECLSPEC_EXPORT
1774  __TNSRegisterNotificationCallback(
1775       IN  PVOID      DeviceHandle,
1776       IN  PVOID      pCallBack,
1777       IN  PVOID      SysParm1,
1778       IN  PVOID      SysParm2,
1779       IN  PVOID      SysParm3)
1780  //
1781  // Description:
1782  //
1783  // Environment:
1784  //
1785  // Return Value:
1786  //
1787  //--
1788  //*************************************************************
1789  {
1790       return TNS_STATUS_NOT_IMPLEMENTED;
1791  }
1792
1793
```

```
1794  //************************************************************************
1795  //-+
1796  TNS_STATUS
1797  DECLSPEC_EXPORT
1798  __TNSDeRegisterNotificationCallback(
1799      IN  PVOID        DeviceHandle,
1800      IN  PVOID        pCallBack)
1801  //
1802  // Description:
1803  //
1804  // Environment:
1805  //
1806  // Return Value:
1807  //
1808  //-
1809  //************************************************************************
1810  {
1811      return TNS_STATUS_NOT_IMPLEMENTED;
1812  }
1813
1814
1815  //************************************************************************
1816  //-+
1817  TNSCPUID
1818  DECLSPEC_EXPORT
1819  __TNSWhoAmI(
1820      IN  PVOID        DeviceHandle)
1821  //
1822  // Description:
1823  //
1824  // Environment:
1825  //
1826  // Return Value:
1827  //
1828  //-
1829  //************************************************************************
1830  {
1831      return 0;
1832  }
1833
1834  //************************************************************************
1835  //-+
1836  TNSCOUNTER
1837  DECLSPEC_EXPORT
1838  __TNSReadOrdinalCounter(
1839      IN  PVOID        DeviceHandle)
1840  //
1841  // Description:
1842  //
1843  // Environment:
1844  //
1845  // Return Value:
1846  //
1847  //-
1848  //************************************************************************
1849  {
1850      return 0;
1851  }
1852
1853
1854  //************************************************************************
1855  //-+
1856  TNS_STATUS
1857  DECLSPEC_EXPORT
1858  __TNSAllocateSharedMemory(
1859      IN      PVOID        DeviceHandle,
1860      IN      TNSKEY       Key,
1861      IN      TNSMEMFLAGS  Flags,
1862      IN      TNSMEMSIZE   Size,
1863      IN OUT  PVOID        *ppBuffer)
1864  //
1865  // Description:
1866  //
1867  // Environment:
1868  //
1869  // Return Value:
1870  //
1871  //-
1872  //************************************************************************
1873  {
1874      return TNS_STATUS_NOT_IMPLEMENTED;
1875  }
```

```
1876
1877  //*************************************************************
1878  //--+
1879  TNS_STATUS
1880  DECLSPEC_EXPORT
1881  __TNSFreeSharedMemory(
1882      IN  PVOID       DeviceHandle,
1883      IN  TNSKEY      Key,
1884      IN  PVOID       Ptr,
1885      IN  TNSMEMSIZE  Size)
1886  //.
1887  //:Description:
1888  //.
1889  //:Environment:
1890  //.
1891  //:Return Value:
1892  //.
1893  //==
1894  //*************************************************************
1895  {
1896      return TNS_STATUS_NOT_IMPLEMENTED;
1897  }
1898
1899  //*************************************************************
1900  //--+
1901  TNS_STATUS
1902  DECLSPEC_EXPORT
1903  __TNSReadSharedMemory(
1904      IN  PVOID       DeviceHandle,
1905      IN  PVOID       pSharedMemoryAddress,
1906      IN  ULONG       Length,
1907      IN  PVOID       pBuffer)
1908  //.
1909  //:Description:
1910  //.
1911  //:Environment:
1912  //.
1913  //:Return Value:
1914  //.
1915  //==
1916  //*************************************************************
1917  {
1918      return TNS_STATUS_NOT_IMPLEMENTED;
1919  }
1920
1921
1922  //*************************************************************
1923  //--+
1924  TNS_STATUS
1925  DECLSPEC_EXPORT
1926  __TNSWriteSharedMemory(
1927      IN  PVOID       DeviceHandle,
1928      IN  PVOID       pSharedMemoryAddress,
1929      IN  ULONG       Length,
1930      IN  PVOID       pBuffer)
1931  //.
1932  //:Description:
1933  //.
1934  //:Environment:
1935  //.
1936  //:Return Value:
1937  //.
1938  //==
1939  //*************************************************************
1940  {
1941      return TNS_STATUS_NOT_IMPLEMENTED;
1942  }
1943
1944  //*************************************************************
1945  //--+
1946  TNS_STATUS
1947  DECLSPEC_EXPORT
1948  __TNSDmaReadSharedMemory(
1949      IN  PVOID       DeviceHandle,
1950      IN  PVOID       pSharedMemoryAddress,
1951      IN  ULONG       Length,
1952      IN  PVOID       pBuffer,
1953      IN  PVOID       pCallback,
1954      IN  PVOID       DMAReadCompleteComtext1,
1955      IN  PVOID       DMAReadCompleteComtext2)
1956  //.
1957  //:Description:
```

```
1958 //:
1959 //:Environment:
1960 //:
1961 //:Return Value:
1962 //:
1963 //:-
1964 //:***************************************************************************
1965 {
1966     return TNS_STATUS_NOT_IMPLEMENTED;
1967 }
1968
1969 //:***************************************************************************
1970 //:-
1971 TNS_STATUS
1972 DECLSPEC_EXPORT
1973 __TNSDmaWriteSharedMemory(
1974     IN  PVOID       DeviceHandle,
1975     IN  PVOID       pSharedMemoryAddress,
1976     IN  ULONG       Length,
1977     IN  PVOID       pBuffer,
1978     IN  PVOID       pCallback,
1979     IN  PVOID       DMAWriteCompleteComtext1,
1980     IN  PVOID       DMAWriteCompleteComtext2)
1981 //:
1982 //:Description:
1983 //:
1984 //:Environment:
1985 //:
1986 //:Return Value:
1987 //:
1988 //:-
1989 //:***************************************************************************
1990 {
1991     return TNS_STATUS_NOT_IMPLEMENTED;
1992 }
1993
1994 //:***************************************************************************
1995 //:-
1996 TNS_STATUS
1997 DECLSPEC_EXPORT
1998 __TNSAllocateWorkQueue(
1999     IN      PVOID       DeviceHandle,
2000     IN      TNSKEY      Key,
2001     IN      PULONG      pQueueLength,
2002     IN OUT  PTNSQUEUE   *ppTNSQueue)
2003 //:
2004 //:Description:
2005 //:
2006 //:Environment:
2007 //:
2008 //:Return Value:
2009 //:
2010 //:-
2011 //:***************************************************************************
2012 {
2013     return TNS_STATUS_NOT_IMPLEMENTED;
2014 }
2015
2016
2017 //:***************************************************************************
2018 //:-
2019 TNS_STATUS
2020 DECLSPEC_EXPORT
2021 __TNSFreeWorkQueue(
2022     IN      PVOID       DeviceHandle,
2023     IN      TNSKEY      Key,
2024     IN      PTNSQUEUE   pTNSQueue)
2025 //:
2026 //:Description:
2027 //:
2028 //:Environment:
2029 //:
2030 //:Return Value:
2031 //:
2032 //:-
2033 //:***************************************************************************
2034 {
2035     return TNS_STATUS_NOT_IMPLEMENTED;
2036 }
2037
2038 //:***************************************************************************
2039 //:-
```

```
2040 TNS_STATUS
2041 DECLSPEC_EXPORT
2042 __TNSInterlockedEnqueueToDoP(
2043     IN      PVOID       DeviceHandle,
2044     IN      PTNSQUEUE   pTNSQueue,
2045     IN      PVOID       pItem,
2046     IN      ULONG       Length)
2047 //
2048 //  Description:
2049 //
2050 //  Environment:
2051 //
2052 //  Return Value:
2053 //
2054 //
2055 //******************************************************
2056 {
2057     return TNS_STATUS_NOT_IMPLEMENTED;
2058 }
2059
2060
2061 //******************************************************
2062 //
2063 TNS_STATUS
2064 DECLSPEC_EXPORT
2065 __TNSInterlockedDequeueToDoP(
2066     IN      PVOID       DeviceHandle,
2067     IN      PTNSQUEUE   pTNSQueue,
2068     IN      PVOID       pItem,
2069     IN      PULONG      pLength)
2070 //
2071 //  Description:
2072 //
2073 //  Environment:
2074 //
2075 //  Return Value:
2076 //
2077 //
2078 //******************************************************
2079 {
2080     return TNS_STATUS_NOT_IMPLEMENTED;
2081 }
2082
2083 //******************************************************
2084 //
2085 TNS_STATUS
2086 DECLSPEC_EXPORT
2087 __TNSQueryQLengthP(
2088     IN      PVOID       DeviceHandle,
2089     IN      PTNSQUEUE   pTNSQueue,
2090     IN      PULONG      pLength)
2091 //
2092 //  Description:
2093 //
2094 //  Environment:
2095 //
2096 //  Return Value:
2097 //
2098 //
2099 //******************************************************
2100 {
2101     return TNS_STATUS_NOT_IMPLEMENTED;
2102 }
2103
2104
2105 //******************************************************
2106 //
2107 TNS_STATUS
2108 DECLSPEC_EXPORT
2109 __TNSQueueHeadP(
2110     IN      PVOID       DeviceHandle,
2111     IN      PTNSQUEUE   pTNSQueue,
2112     IN OUT  PTNSQUEUE   *ppTNSQueue)
2113 //
2114 //  Description:
2115 //
2116 //  Environment:
2117 //
2118 //  Return Value:
2119 //
2120 //
2121 //******************************************************
```

```
2122 {
2123     return TNS_STATUS_NOT_IMPLEMENTED;
2124 }
2125
2126
2127 //*****************************************************************
2128 //+
2129 TNS_STATUS
2130 DECLSPEC_EXPORT
2131 __TNSQueueTailP(
2132     IN        PVOID        DeviceHandle,
2133     IN        PTNSQUEUE    pTNSQueue,
2134     IN OUT    PTNSQUEUE    *ppTNSQueue)
2135 //
2136 //  Description:
2137 //
2138 //  Environment:
2139 //
2140 //  Return Value:
2141 //
2142 //--
2143 //*****************************************************************
2144 {
2145     return TNS_STATUS_NOT_IMPLEMENTED;
2146 }
2147
2148
2149 //*****************************************************************
2150 //+
2151 TNS_STATUS
2152 DECLSPEC_EXPORT
2153 __TNSQueuePayloadP(
2154     IN        PVOID        DeviceHandle,
2155     IN        PTNSQUEUE    pTNSQueue,
2156     IN        PVOID        pItem,
2157     IN        PULONG       pLength)
2158 //
2159 //  Description:
2160 //
2161 //  Environment:
2162 //
2163 //  Return Value:
2164 //
2165 //
2166 //*****************************************************************
2167 {
2168     return TNS_STATUS_NOT_IMPLEMENTED;
2169 }
2170
2171
2172 //*****************************************************************
2173 //+
2174 TNS_STATUS
2175 DECLSPEC_EXPORT
2176 __TNSQueueNextP(
2177     IN        PVOID        DeviceHandle,
2178     IN        PTNSQUEUE    pTNSQueue,
2179     IN OUT    PTNSQUEUE    *ppTNSQueue)
2180 //
2181 //  Description:
2182 //
2183 //  Environment:
2184 //
2185 //  Return Value:
2186 //
2187 //--
2188 //*****************************************************************
2189 {
2190     return TNS_STATUS_NOT_IMPLEMENTED;
2191 }
2192
2193 //*****************************************************************
2194 //+
2195 TNS_STATUS
2196 DECLSPEC_EXPORT
2197 __TNSInterlockedInsertQueueItemP(
2198     IN        PVOID        DeviceHandle,
2199     IN        PTNSQUEUE    pTNSQueue,
2200     IN        PTNSQUEUE    pTNSQueueInsert)
2201 //
2202 //  Description:
2203 //
```

```
2204  // Environment:
2205  //
2206  // Return Value:
2207  //
2208  //
2209  //*****************************************************************
2210  {
2211      return TNS_STATUS_NOT_IMPLEMENTED;
2212  }
2213
2214
2215  //*****************************************************************
2216  //
2217  TNS_STATUS
2218  DECLSPEC_EXPORT
2219  __TNSInterlockedDeleteQueueItemP(
2220      IN      PVOID         DeviceHandle,
2221      IN      PTNSQUEUE     pTNSQueue,
2222      IN      PTNSQUEUE     pTNSQueueDelete)
2223  //
2224  // Description:
2225  //
2226  // Environment:
2227  //
2228  // Return Value:
2229  //
2230  //
2231  //*****************************************************************
2232  {
2233      return TNS_STATUS_NOT_IMPLEMENTED;
2234  }
2235
2236  //*****************************************************************
2237  //
2238  TNS_STATUS
2239  DECLSPEC_EXPORT
2240  __TNSQueueItemInfoP(
2241      IN      PVOID          DeviceHandle,
2242      IN      PTNSQUEUE      pTNSQueue,
2243      IN      PTNSQUEUEINFO  pTNSQueueInfo)
2244  //
2245  // Description:
2246  //
2247  // Environment:
2248  //
2249  // Return Value:
2250  //
2251  //
2252  //*****************************************************************
2253  {
2254      return TNS_STATUS_NOT_IMPLEMENTED;
2255  }
2256
2257
2258  //*****************************************************************
2259  //
2260  TNS_STATUS
2261  DECLSPEC_EXPORT
2262  __TNSGetFirstDeviceInstance(
2263      PVOID    *ppDeviceInstance)
2264  //
2265  // Description:
2266  //
2267  // Environment:
2268  //
2269  // Return Value:
2270  //
2271  //
2272  //*****************************************************************
2273  {
2274      return TNS_STATUS_NOT_IMPLEMENTED;
2275  }
2276
2277  //*****************************************************************
2278  //
2279  TNS_STATUS
2280  DECLSPEC_EXPORT
2281  __TNSGetNextDeviceInstance(
2282      PVOID    pDeviceInstance,
2283      PVOID    *ppDeviceInstance)
2284  //
2285  // Description:
```

```
2286 //
2287 // Environment:
2288 //
2289 // Return Value:
2290 //
2291 //
2292 //*********************************************************************
2293 {
2294     return TNS_STATUS_NOT_IMPLEMENTED;
2295 }
2296
2297
2298
2299 //*********************************************************************
2300 //
2301 ULONG
2302 DECLSPEC_EXPORT
2303 _TNS_GET_SMN_STATISTICS(
2304     IN        PVOID        DeviceHandle,
2305     IN OUT    PSTATISTICS  pStatistics,
2306     IN OUT    PULONG       pStatsStructSize,
2307     IN OUT    pMPSTATS     pMpStats,
2308     IN OUT    PULONG       pMpStatsSize)
2309 //
2310 // Description:
2311 //
2312 // Environment:
2313 //
2314 // Return Value:
2315 //
2316 //
2317 //*********************************************************************
2318 {
2319     PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2320     NTSTATUS    Status;
2321     KIRQL    OldIrql;
2322     PNDIS_PACKET MyPacket;
2323     ULONG PacketLength;
2324     PTNSPacketQueryStats pTnsBuffer;
2325     PLIST_ENTRY clientRequest;
2326     PREQUEST_DATA pClientRequestData;
2327     ULONG requestTag;
2328     ULONG retries=0;
2329     int noreply = TRUE;
2330     ULONG returnRequestTag;
2331
2332     //
2333     // hack hack - we really wanna use the device context given up
2334     // by the caller.
2335     //
2336     pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2337
2338     //
2339     // Raise IRQL to prevent task swapping while we complete processing
2340     // for this packet.
2341     //
2342     KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);
2343
2344     //
2345     // Make sure driver has been initialized properly (this is
2346     // an assertion: this case should never happen).
2347     //
2348     //
2349     // hack hack work on error handling
2350     //
2351     if (!pAdapter->TNSDriverInitialized) {
2352         BreakPoint();
2353         KeLowerIrql(OldIrql);
2354         return 0;
2355     }
2356
2357     //
2358     // compute packet length, based on request, and
2359     // set the variable accordingly (the packet structure length
2360     // will get set according to this variable).
2361     //
2362
2363     PacketLength = TNS_PACKET_SIZE(TNSPacketQueryStats);
2364
2365     requestTag = TNSGetRequestTag();
2366
2367     while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {
```
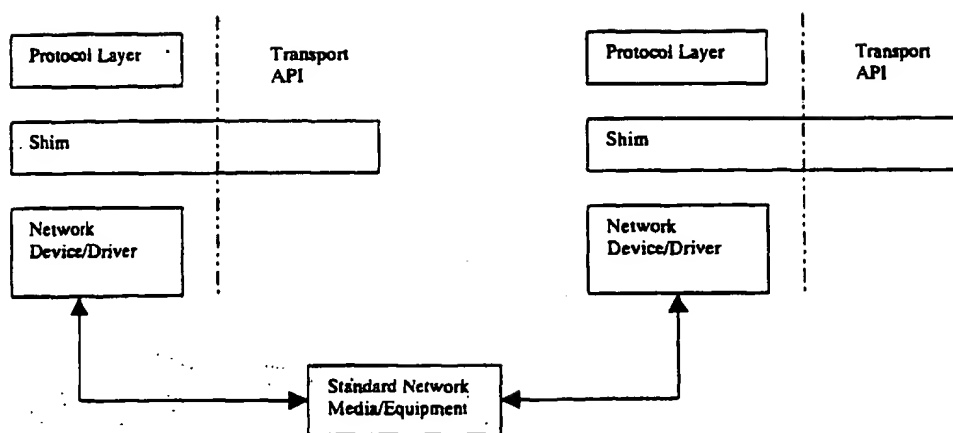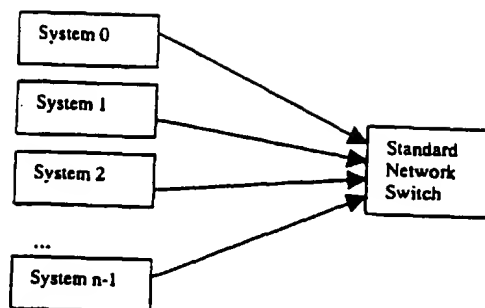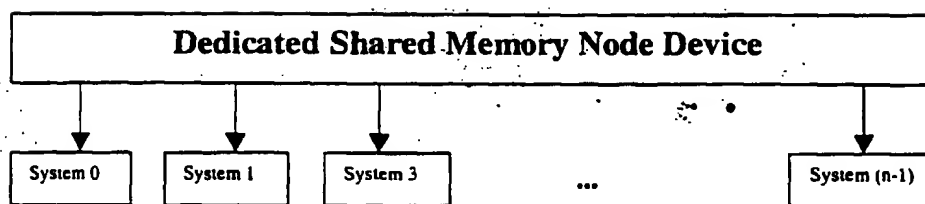
FIGURE 1

FIGURE 2



FIGURE 3

```
2368
2369              Status = TNSInitializeClientNodeSendPacket(pAdapter,
2370                  &MyPacket,
2371                  &pTnsBuffer,
2372                  PacketLength);
2373
2374              //
2375              // Fill in relevent packet information here
2376              //
2377              pTnsBuffer->TNSCommandReply = wswap(TNS_QUERY_STATS);
2378
2379              pTnsBuffer->RequestTag = dwswap(requestTag);
2380              pTnsBuffer->RequestStartTSC = rdtsc();
2381
2382              if (NT_SUCCESS(Status)) {
2383                  PLIST_ENTRY wrkrRequest;
2384                  PREQUEST_DATA pWrkrRequestData;
2385                  LARGE_INTEGER queueWait;
2386                  int timeout = FALSE;
2387                  int ltimeout = FALSE;
2388                  int timeoutcount = 0;
2389
2390                  //
2391                  // Flush the read reply queue. In case a different request timed out,
2392                  // and it actually shows up, we need to flush the queue for
2393                  // subsequent requests.
2394                  //
2395                  TNSFlushReadReplyQueue(pAdapter);
2396
2397                  //
2398                  // Send request packet to SMN
2399                  //
2400                  TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
2401
2402                  //
2403                  // This is a read operation, so we expect a response.
2404                  // Block waiting for the response from the SMN.
2405                  //
2406                  // this is 100m secs.
2407                  //
2408                  queueWait.QuadPart = -(1000000);
2409
2410                  Status = KeWaitForSingleObject(
2411                      (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
2412                      Executive,
2413                      KernelMode,
2414                      FALSE,
2415                      &queueWait);
2416
2417                  if (Status != STATUS_TIMEOUT) {
2418                      PTNSPacketQueryStatsReply pTnsPacketQueryStatsReply;
2419
2420                      clientRequest = ExInterlockedRemoveHeadList(
2421                          &pAdapter->ClientWorkerListEntry,
2422                          &pAdapter->ClientWorkerListSpinLock);
2423
2424                      MyAssert(clientRequest != NULL);
2425
2426                      pClientRequestData = CONTAINING_RECORD(clientRequest,
2427                              REQUEST_DATA,
2428                              Linkage);
2429
2430                      MyAssert(pClientRequestData != NULL);
2431
2432                      pTnsPacketQueryStatsReply = (PTNSPacketQueryStatsReply) &pClientRequestData->TnsPacke
2433
2434                      returnRequestTag    = dwswap(pTnsPacketQueryStatsReply->RequestTag);
2435                      MyAssert(returnRequestTag == requestTag);
2436
2437                      if (returnRequestTag == requestTag) {
2438                          noreply = FALSE;
2439                          RtlCopyMemory(pStatistics, &pTnsPacketQueryStatsReply->TnsNodeStatistics, sizeof(
 -2 ISTICS) );
2440                          RtlCopyMemory(pMpStats, &pTnsPacketQueryStatsReply->MpStats, sizeof(MPSTATS) );
2441                      }
2442                      //
2443                      // Recycle the queue object
2444                      //
2445                      ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
2446                          &pClientRequestData->Linkage,
2447                          &pAdapter->ListEntryPoolLock);
2448                  } else {
```

84

```
2449                        //
2450                        //do something useful ?
2451                        //
2452                  }
2453             }
2454        }
2455
2456        KeLowerIrql(OldIrql);
2457
2458        if (noreply == TRUE) {
2459             //
2460             //Throw an exception to our client
2461             //
2462             //TODO
2463        }
2464
2465        return 0;
2466 }
2467
2468
2469
2470 //*****************************************************************************
2471 //-+
2472 ULONG
2473 DECLSPEC_EXPORT
2474 __TNS_GET_SMN_STATISTICS_BY_NODEID(
2475        IN       PVOID        DeviceHandle,
2476        IN       ULONG        NodeID,
2477        IN OUT   PSTATISTICS  pStatistics,
2478        IN OUT   PULONG       pStatsStructSize,
2479        IN OUT   pMPSTATS     pMpStats,
2480        IN OUT   PULONG       pMpStatsSize)
2481 //
2482 //Description:
2483 //
2484 //Environment:
2485 //
2486 //Return Value:
2487 //
2488 //
2489 //*****************************************************************************
2490 {
2491        PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2492        NTSTATUS     Status;
2493        KIRQL    OldIrql;
2494        PNDIS_PACKET MyPacket;
2495        ULONG PacketLength;
2496        PTNSPacketQueryStats pTnsBuffer;
2497        PLIST_ENTRY clientRequest;
2498        PREQUEST_DATA pClientRequestData;
2499        ULONG requestTag;
2500        ULONG retries=0;
2501        int noreply = TRUE;
2502        ULONG returnRequestTag;
2503        ULONG retValue = 0;
2504
2505        //
2506        //hack hack. We really wanna use the device context given up
2507        //by the caller.
2508        //
2509        pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2510
2511        if (TNSSharedMemoryNodeEmulation) {
2512
2513             //
2514             //Find index into SMN node info table, make sure
2515             //it's valid.
2516             //
2517             if (NodeID < MAX_TEAM_NODES) {
2518                  if (pAdapter->TeamNodeTable[NodeID].LocationSet == 0) {
2519                       return 0;
2520                  }
2521             } else {
2522                  return 0;
2523             }
2524
2525             //
2526             //Raise IRQL to prevent task swapping while we complete processing
2527             //for this packet.
2528             //
2529             KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);
2530
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c**                    **Page 32 of 39**

```
2531    //
2532    // Make sure driver has been initialized properly (this is
2533    // an assertion, this case should never happen).
2534    //
2535    //
2536    // hack hack work on error handling
2537    //
2538    if (!pAdapter->TNSDriverInitialized) {
2539        BreakPoint();
2540        KeLowerIrql(OldIrql);
2541        return 0;
2542    }
2543
2544    //
2545    // compute packet length, based on request, and
2546    // set the variable accordingly (the packet structure length
2547    // will get set according to this variable).
2548    //
2549
2550    PacketLength = TNS_PACKET_SIZE(TNSPacketQueryStats);
2551
2552    requestTag = TNSGetRequestTag();
2553
2554    while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {
2555
2556        Status = TNSInitializeClientNodeSendPacket(pAdapter,
2557            &MyPacket,
2558            &pTnsBuffer,
2559            PacketLength);
2560
2561        //
2562        // set directed packet address by node id
2563        //
2564        RtlCopyMemory(
2565            pTnsBuffer->MACDstAddress,
2566            pAdapter->TeamNodeTable[NodeID].TNMacAddress,
2567            ETH_ADDRESS_LEN);
2568
2569        //
2570        // fill in relevant packet information here
2571        //
2572        pTnsBuffer->TNSCommandReply = wswap(TNS_QUERY_STATS);
2573
2574        pTnsBuffer->RequestTag = dwswap(requestTag);
2575        pTnsBuffer->RequestStartTSC = rdtsc();
2576
2577        if (NT_SUCCESS(Status)) {
2578            PLIST_ENTRY wrkrRequest;
2579            PREQUEST_DATA pWrkrRequestData;
2580            LARGE_INTEGER queueWait;
2581            int timeout = FALSE;
2582            int ltimeout = FALSE;
2583            int timeoutcount = 0;
2584
2585            //
2586            // flush the read reply queue. In case a different request timed out,
2587            // and it actually shows up, we need to flush the queue for
2588            // subsequent requests.
2589            //
2590            TNSFlushReadReplyQueue(pAdapter);
2591
2592            //
2593            // send request packet to SMN
2594            //
2595            TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
2596
2597            //
2598            // this is a read operation, so we expect a response.
2599            // block waiting for the response from the SMN.
2600            //
2601            // this is 100m secs.
2602            //
2603            queueWait.QuadPart = -(1000000);
2604
2605            Status = KeWaitForSingleObject(
2606                (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
2607                Executive,
2608                KernelMode,
2609                FALSE,
2610                &queueWait);
2611
2612            if (Status != STATUS_TIMEOUT) {
```

```
2613                       PTNSPacketQueryStatsReply pTnsPacketQueryStatsReply;
2614
2615                       clientRequest = ExInterlockedRemoveHeadList(
2616                           &pAdapter->ClientWorkerListEntry,
2617                           &pAdapter->ClientWorkerListSpinLock);
2618
2619                       MyAssert(clientRequest != NULL);
2620
2621                       pClientRequestData = CONTAINING_RECORD(clientRequest,
2622                                   REQUEST_DATA,
2623                                   Linkage);
2624
2625                       MyAssert(pClientRequestData != NULL);
2626
2627                       pTnsPacketQueryStatsReply = (PTNSPacketQueryStatsReply) &pClientRequestData->TnsPacke
  -2 t;
2628
2629                       returnRequestTag     = dwswap(pTnsPacketQueryStatsReply->RequestTag);
2630                       //MyAssert(returnRequestTag == requestTag);
2631
2632                       if (returnRequestTag == requestTag) {
2633                           noreply = FALSE;
2634                           RtlCopyMemory(pStatistics, &pTnsPacketQueryStatsReply->TnsNodeStatistics, sizeof(
  -2 STATISTICS) );
2635                           RtlCopyMemory(pMpStats, &pTnsPacketQueryStatsReply->MpStats, sizeof(MPSTATS) );
2636                           retValue = 1;
2637                       }
2638                       //
2639                       // Recycle the queue object
2640                       //
2641                       ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
2642                                   &pClientRequestData->Linkage,
2643                                   &pAdapter->ListEntryPoolLock);
2644                   } else {
2645                       //
2646                       // do something useful ?
2647                       //
2648                   }
2649               }
2650           }
2651
2652       KeLowerIrql(OldIrql);
2653
2654       if (noreply == TRUE) {
2655           //
2656           // Throw an exception to our client
2657           //
2658           // TODO
2659       }
2660   } else {
2661   }
2662
2663   return 0;
2664 }
2665
2666 //************************************************************************
2667 //
2668 ULONG
2669 DECLSPEC_EXPORT
2670 _TNS_GET_SMN_INFORMATION(
2671     IN      PVOID       DeviceHandle,
2672     IN OUT  unsigned char *pMacAddress,
2673     IN OUT  unsigned char *pNodeName,
2674     IN OUT  unsigned long *pSharedMemorySize)
2675 //
2676 // Description:
2677 //
2678 // Environment:
2679 //
2680 // Return Value:
2681 //
2682 //
2683 //************************************************************************
2684 {
2685     PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2686     pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2687
2688     RtlCopyMemory(pMacAddress, &pAdapter->SMNMacAddress, HARDWARE_ADDRESS_LENGTH);
2689     RtlCopyMemory(pNodeName, &pAdapter->SMNMachineName, 16);
2690     *pSharedMemorySize = pAdapter->TNSSharedMemorySize;
2691     return 0;
2692 }
```

```
2693
2694  //****************************************************************
2695  //--+
2696  ULONG
2697  DECLSPEC_EXPORT
2698  __TNS_GET_NODE_INFORMATION(
2699        IN       PVOID       DeviceHandle,
2700        IN OUT  unsigned char *pMacAddress,
2701        IN OUT  unsigned char *pNodeName,
2702        IN OUT  unsigned int  *pNodeID)
2703  //
2704  // Description:
2705  //
2706  // Environment:
2707  //
2708  // Return Value:
2709  //
2710  //--
2711  //****************************************************************
2712  {
2713        PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2714        pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2715
2716        RtlCopyMemory(pMacAddress, &pAdapter->LowerMPMacAddress, HARDWARE_ADDRESS_LENGTH);
2717        RtlCopyMemory(pNodeName, &pAdapter->LocalComputerName, 16);
2718        *pNodeID = pAdapter->TNSClientNodeID;
2719        return 0;
2720  }
2721
2722  //****************************************************************
2723  //--+
2724  ULONG
2725  DECLSPEC_EXPORT
2726  __TNS_CLEAR_NODE_STATISTICS(
2727        IN       PVOID       DeviceHandle)
2728  //
2729  // Description:
2730  //
2731  // Environment:
2732  //
2733  // Return Value:
2734  //
2735  //--
2736  //****************************************************************
2737  {
2738        PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2739        pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2740
2741        RtlZeroMemory(&pAdapter->MyStats, sizeof(STATISTICS));
2742        RtlZeroMemory(&pAdapter->mpStats, sizeof(MPSTATS));
2743        GetProcessorSpeed(pAdapter);
2744        return 0;
2745  }
2746
2747
2748
2749  //****************************************************************
2750  //--
2751  ULONG
2752  DECLSPEC_EXPORT
2753  __TNS_GET_SMN_TABLE_INFO(
2754        IN       PVOID       DeviceHandle,
2755        IN OUT  pSMNTableInfo pSMNInfo)
2756  //
2757  // Description:
2758  //
2759  // Environment:
2760  //
2761  // Return Value:
2762  //
2763  //--
2764  //****************************************************************
2765  {
2766        PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2767        ULONG retValue=0;
2768        int i,j;
2769
2770        pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2771
2772        if (TNSSharedMemoryNodeEmulation) {
2773              //
2774              // Return true if we are an SMN
```

```
2775        //
2776        retValue - 1;
2777        for (i=0; i<MAX_TEAM_NODES; i++) {
2778            pSMNInfo->LocationSet = pAdapter->TeamNodeTable[i].LocationSet;
2779            for (j=0; j<6; j++) {
2780                pSMNInfo->MacAddress[j] = pAdapter->TeamNodeTable[i].TNMacAddress[j];
2781            }
2782            for (j=0; j<MAX_COMPUTER_NAME_SIZE; j++) {
2783                pSMNInfo->ComputerName[j] = pAdapter->TeamNodeTable[i].TNComputerName[j];
2784            }
2785            pSMNInfo->NodeID = pAdapter->TeamNodeTable[i].TNNodeID;
2786            pSMNInfo++;
2787        }
2788    }
2789
2790    return retValue;
2791 }
2792
2793 //*********************************************************************
2794 //
2795 ULONG
2796 DECLSPEC_EXPORT
2797 __TNS_CLEAR_SMN_STATISTICS(
2798        IN      PVOID         DeviceHandle)
2799 //
2800 // Description:
2801 //
2802 // Environment:
2803 //
2804 // Return Value:
2805 //
2806 //
2807 //*********************************************************************
2808 {
2809    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2810    NTSTATUS    Status;
2811    KIRQL   OldIrql;
2812    PNDIS_PACKET MyPacket;
2813    ULONG PacketLength;
2814    PTNSPacketClearStats pTnsBuffer;
2815    PLIST_ENTRY clientRequest;
2816    PREQUEST_DATA pClientRequestData;
2817    ULONG requestTag;
2818    ULONG retries=0;
2819    int noreply - TRUE;
2820    ULONG returnRequestTag;
2821
2822    //
2823    // hack hack we really wanna use the device context given up
2824    // by the caller.
2825    //
2826    pAdapter - CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2827
2828    //
2829    // Raise IRQL to prevent task swapping while we complete processing
2830    // for this packet.
2831    //
2832    KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);
2833
2834    //
2835    // Make sure driver has been initialized properly (this is
2836    // an assertion, this case should never happen).
2837    //
2838    //
2839    // hack hack work on error handling
2840    //
2841    if (!pAdapter->TNSDriverInitialized) {
2842        BreakPoint();
2843        KeLowerIrql(OldIrql);
2844        return 0;
2845    }
2846
2847    //
2848    // compute packet length, based on request, and
2849    // set the variable accordingly (the packet structure length
2850    // will get set according to this variable).
2851    //
2852
2853    PacketLength = TNS_PACKET_SIZE(TNSPacketClearStats);
2854
2855    requestTag - TNSGetRequestTag();
2856
```

```
2857        while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {
2858
2859            Status = TNSInitializeClientNodeSendPacket(pAdapter,
2860                &MyPacket,
2861                &pTnsBuffer,
2862                PacketLength);
2863
2864            //
2865            // Fill in event packet information here
2866            //
2867            pTnsBuffer->TNSCommandReply = wswap(TNS_CLEAR_STATS);
2868
2869            pTnsBuffer->RequestTag = dwswap(requestTag);
2870            pTnsBuffer->RequestStartTSC = rdtsc();
2871
2872            if (NT_SUCCESS(Status)) {
2873                PLIST_ENTRY wrkrRequest;
2874                PREQUEST_DATA pWrkrRequestData;
2875                LARGE_INTEGER queueWait;
2876                int timeout = FALSE;
2877                int ltimeout = FALSE;
2878                int timeoutcount = 0;
2879
2880                //
2881                // Send request packet to SMN
2882                //
2883                TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
2884            }
2885        }
2886        KeLowerIrql(OldIrql);
2887
2888        return 0;
2889 }
2890
2891
2892
2893 //****************************************************************************
2894 //
2895 ULONG
2896 DECLSPEC_EXPORT
2897 __TNS_GET_NODE_STATISTICS(
2898     IN        PVOID      DeviceHandle,
2899     IN OUT    PSTATISTICS pStatistics,
2900     IN OUT    PULONG      pStatsStructSize,
2901     IN OUT    pMPSTATS     pMpStats,
2902     IN OUT    PULONG      pMpStatsSize)
2903 //
2904 // Description:
2905 //
2906 // Environment:
2907 //
2908 // Return Value:
2909 //
2910 //
2911 //****************************************************************************
2912 {
2913     PADAPTER pAdapter = (PADAPTER) DeviceHandle;
2914     NDIS_STATUS NdisStatus;
2915     //
2916     // Hack Hack. We really wanna use the device context given us
2917     // by the caller.
2918     //
2919     pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);
2920
2921     MyAssert(pStatsStructSize);
2922     MyAssert(pMpStatsSize);
2923
2924     if ( (*pStatsStructSize >= sizeof (STATISTICS)) && (pStatistics) ) {
2925         RtlCopyMemory(pStatistics, &pAdapter->MyStats, sizeof(STATISTICS) );
2926     } else {
2927         *pStatsStructSize = sizeof (STATISTICS);
2928         return 0;
2929     }
2930     if( (*pMpStatsSize >= sizeof (MPSTATS)) && (pMpStats) ) {
2931         TnsGetNICStats(pAdapter, pMpStats);
2932     } else {
2933         *pMpStatsSize = sizeof (MPSTATS) ;
2934         return 0;
2935     }
2936
2937     return 1;
2938 }
```

```
2939
2940
2941
2942
2943 unsigned char zerobuffer[6] = { 0, 0, 0, 0, 0, 0 };
2944
2945 VOID
2946 TNSSendPackets(
2947     IN  NDIS_HANDLE          NdisBindingHandle,
2948     IN  PPNDIS_PACKET        PacketArray,
2949     IN  UINT                 NumberOfPackets)
2950 {
2951     UINT PhysBufferCount, BufferCount, PacketLength;
2952     PNDIS_BUFFER FirstBuffer, NextBuffer;
2953     PUCHAR va;
2954     UINT bufferLength;
2955     unsigned short *pEtherType;
2956     unsigned int i,j;
2957     NDIS_STATUS Status;
2958     int Found;
2959
2960
2961     for (i=0; i<NumberOfPackets; i++) {
2962
2963 #ifdef DBG
2964         NdisQueryPacket(PacketArray[i], &PhysBufferCount, &BufferCount, &FirstBuffer, &PacketLength);
2965
2966         NextBuffer = FirstBuffer;
2967         for (j=0; NextBuffer!= NULL; j++) {
2968             NdisQueryBuffer(NextBuffer, &va, &bufferLength);
2969
2970             if (j==0) {
2971                 MyAssert(bufferLength != 0);
2972                 if (bufferLength >= 14) {
2973                     pEtherType = (unsigned short *)&va[12];
2974                     MyAssert (wswap(*pEtherType) == TNS_EMULATION_ETHERTYPE);
2975                     MyAssert (RtlCompareMemory(va, zerobuffer, 6) != 6);
2976                     MyAssert (RtlCompareMemory(&va[6], zerobuffer, 6) != 6);
2977                 }
2978             }
2979             NdisGetNextBuffer(NextBuffer, &NextBuffer);
2980         }
2981 #endif
2982         NdisSend(&Status, NdisBindingHandle, PacketArray[i]);
2983
2984 #ifdef DBG
2985         switch (Status) {
2986             case NDIS_STATUS_SUCCESS:
2987                 break;
2988             case NDIS_STATUS_PENDING:
2989                 break;
2990             case NDIS_STATUS_INVALID_PACKET:
2991                 MyAssert(0);
2992                 break;
2993             case NDIS_STATUS_CLOSING:
2994                 MyAssert(0);
2995                 break;
2996             case NDIS_STATUS_RESET_IN_PROGRESS:
2997                 MyAssert(0);
2998                 break;
2999             case NDIS_STATUS_FAILURE:
3000                 MyAssert(0);
3001                 break;
3002             default:
3003                 MyAssert(0);
3004                 D((0, "Status => %x, %s\n", Status, GetNDISStatusString(Status, &Found) ));
3005                 break;
3006         }
3007 #endif
3008
3009     }
3010     //(NdisSendPackets(NdisBindingHandle, PacketArray, NumberOfPackets));
3011 }
3012
3013 NDIS_STATUS
3014 TnsGetNICStats(
3015     PADAPTER     pAdapter,
3016     pMPSTATS     pMpStats)
3017 {
3018     NDIS_STATUS NdisStatus;
3019
3020     NdisStatus = MakeLocalNdisRequest(
```

```
3021            pAdapter,
3022            OID_GEN_XMIT_OK,
3023            &pMpStats->XmitOK,
3024            sizeof(ULONG));
3025        if (NdisStatus != NDIS_STATUS_SUCCESS) {
3026            //return NdisStatus;
3027            _asm int 3
3028        }
3029
3030        NdisStatus = MakeLocalNdisRequest(
3031            pAdapter,
3032            OID_GEN_RCV_OK,
3033            &pMpStats->RcvOK,
3034            sizeof(ULONG));
3035        if (NdisStatus != NDIS_STATUS_SUCCESS) {
3036            asm int 3
3037            //return NdisStatus;
3038        }
3039
3040        NdisStatus = MakeLocalNdisRequest(
3041            pAdapter,
3042            OID_GEN_XMIT_ERROR,
3043            &pMpStats->XmitError,
3044            sizeof(ULONG));
3045        if (NdisStatus != NDIS_STATUS_SUCCESS) {
3046            asm int 3
3047            //return NdisStatus;
3048        }
3049
3050        NdisStatus = MakeLocalNdisRequest(
3051            pAdapter,
3052            OID_GEN_RCV_ERROR,
3053            &pMpStats->RcvError,
3054            sizeof(ULONG));
3055        if (NdisStatus != NDIS_STATUS_SUCCESS) {
3056            asm int 3
3057            //return NdisStatus;
3058        }
3059
3060        NdisStatus = MakeLocalNdisRequest(
3061            pAdapter,
3062            OID_GEN_RCV_NO_BUFFER,
3063            &pMpStats->RcvNoBuffer,
3064            sizeof(ULONG));
3065        if (NdisStatus != NDIS_STATUS_SUCCESS) {
3066            asm int 3
3067            //return NdisStatus;
3068        }
3069
3070        NdisStatus = MakeLocalNdisRequest(
3071            pAdapter,
3072            OID_GEN_RCV_CRC_ERROR,
3073            &pMpStats->RcvCrcError,
3074            sizeof(ULONG));
3075        if (NdisStatus != NDIS_STATUS_SUCCESS) {
3076            asm int 3
3077            //return NdisStatus;
3078        }
3079
3080
3081        return NDIS_STATUS_SUCCESS;
3082    }
3083
3084
3085    VOID
3086    TnsAddStatsUlong(
3087        PADAPTER pAdapter,
3088        PLARGE_INTEGER pLi,
3089        ULONG Addend)
3090    {
3091        LARGE_INTEGER AddendPart;
3092
3093        AddendPart.HighPart = 0;
3094        AddendPart.LowPart = Addend;
3095
3096        (void)ExInterlockedAddLargeInteger(pLi, AddendPart, &pAdapter->MyStatsLock);
3097    }
3098
3099    VOID
3100    TnsIncrementStat(
3101        PADAPTER pAdapter,
3102        PLARGE_INTEGER pLi)
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c**                    **Page 39 of 39**

```
3103 {
3104     LARGE_INTEGER Addend;
3105
3106     Addend.QuadPart = 1;
3107
3108     (void)ExInterlockedAddLargeInteger(pLi, Addend, &pAdapter->MyStatsLock);
3109 }
3110
3111 unsigned long _fltused;
3112
3113 void
3114 GetProcessorSpeed(
3115     PADAPTER pAdapter)
3116 {
3117     LARGE_INTEGER qPerfCounter1, qPerfCounter2, qPerfDiff, qPerfFreq;
3118     //double freq;
3119
3120     LARGE_INTEGER qPerfInc = {65536, 0};
3121     LARGE_INTEGER qrdtsc1, qrdtsc2, qrdtscdiff;
3122
3123     qPerfCounter1 = KeQueryPerformanceCounter(&qPerfFreq);
3124
3125     qPerfCounter2.QuadPart = qPerfCounter1.QuadPart + qPerfInc.QuadPart;
3126
3127     qrdtsc1 = rdtsc();
3128     do {
3129         qPerfCounter1 = KeQueryPerformanceCounter(NULL);
3130         qrdtsc2 = rdtsc();
3131     } while (qPerfCounter1.QuadPart < qPerfCounter2.QuadPart) ;
3132
3133     qPerfDiff.QuadPart = qPerfCounter1.QuadPart - (qPerfCounter2.QuadPart - qPerfInc.QuadPart);
3134     qrdtscdiff.QuadPart = qrdtsc2.QuadPart - qrdtsc1.QuadPart;
3135
3136     //freq = (double) qrdtscdiff.LowPart*(double)qPerfFreq.LowPart)/(double)qPerfDiff.LowPart;
3137
3138     pAdapter->MyStats.rdtscDiff = qrdtscdiff.LowPart;
3139     pAdapter->MyStats.perfFreq  = qPerfFreq.LowPart;
3140     pAdapter->MyStats.perfDiff  = qPerfDiff.LowPart;
3141
3142     D((0, "qrdtscdiff.LowPart => %x\n", qrdtscdiff.LowPart    ));
3143     D((0, "qPerfFreq.LowPart  => %x\n", qPerfFreq.LowPart     ));
3144     D((0, "qPerfDiff.LowPart  => %x\n", qPerfDiff.LowPart     ));
3145 }
3146
3147
```

```
  1 ////////////////////////////////////////////////////////////////////
  2 //
  3 // COPYRIGHT:
  4 //    This program is an unpublished work fully protected by the United
  5 //    States copyright laws and is considered a trade secret belonging to
  6 //    Times N Systems, Inc.  To the extent that this work may be
  7 //    considered published, the following notice applies: 1999, Times N
  8 //    Systems, Inc.  Any unauthorized use, reproduction, distribution,
  9 //    display, modification, or disclosure of this program is strictly
 10 //    prohibited.
 11 //
 12 ////////////////////////////////////////////////////////////////////
 13 //*
 14 ////////////////////////////////////////////////////////////////////
 15 // Module:
 16 //    tnsemul.c  Main initialization and support routine module
 17 //    for Times N High Speed Interconnect emulation driver.
 18 //
 19 // Description:
 20 //
 21 // Environment:
 22 //    Windows NT Kernel Mode, Ndis driver models only.
 23 //
 24 // Exports:
 25 //    See Module functions generated by script processing.
 26 //
 27 // Author:
 28 //    Vince Bridgers
 29 //    vinceb@timesn.com
 30 //
 31 ////////////////////////////////////////////////////////////////////
 32
 33 #include "tns.h"
 34 #include "tnsdebug.h"
 35
 36 PADAPTER CurrentAdapter;
 37 ULONG    TNSSharedMemoryNodeEmulation = FALSE;
 38
 39 NDIS_PHYSICAL_ADDRESS HighAddress = NDIS_PHYSICAL_ADDRESS_CONST( -1, -1 );
 40
 41 LIST_ENTRY AdapterList;
 42 NDIS_SPIN_LOCK AdapterListLock;
 43
 44 NDIS_HANDLE ClientProtocolHandle;
 45
 46 NDIS_HANDLE MPWrapperHandle;
 47
 48 NDIS_HANDLE LMDriverHandle;
 49
 50 PDRIVER_OBJECT IMDriverObject;
 51 PDEVICE_OBJECT IMDeviceObject;
 52
 53 CONFIG_DATA ConfigData;
 54
 55 NDIS_STRING IMSymbolicName = NDIS_STRING_CONST("\\DosDevices\\Im");
 56 NDIS_STRING IMDriverName =   NDIS_STRING_CONST("\\Device\\Im" );
 57 NDIS_STRING IMMPName =       NDIS_STRING_CONST("\\Device\\Im" );
 58
 59 DECLARE_STRING( PacketPoolSize );
 60 DECLARE_STRING( DebugLevel );
 61 DECLARE_STRING( DebugMask );
 62 DECLARE_STRING( TNSSMNEmulationMode );
 63
 64 ////////////////////////////////////////////////////////////////////
 65 //
 66 // Function prototypes
 67 //
 68 ////////////////////////////////////////////////////////////////////
 69
 70 NTSTATUS
 71 DriverEntry(
 72     IN PDRIVER_OBJECT DriverObject,
 73     IN PUNICODE_STRING RegistryPath);
 74
 75 STATIC NDIS_STATUS
 76 GetAdapterRegistryData(
 77     PNDIS_STRING IMParamsKey,
 78     PADAPTER pAdapter);
 79
 80 STATIC VOID
 81 ProcessLowerMPOpenAdapter(
 82     IN  PADAPTER pAdapter,
```

```
 83     IN  NDIS_STATUS Status);
 84
 85 STATIC NDIS_STATUS
 86 AllocatePacketPool(
 87    . PADAPTER pAdapter);
 88
 89 STATIC NDIS_STATUS
 90 AllocateReceiveBufferPools(
 91     PADAPTER pAdapter);
 92
 93 STATIC ULONG
 94 ReadSingleParameter(
 95     IN NDIS_HANDLE ParametersHandle,
 96     IN PWCHAR ValueName,
 97     IN ULONG DefaultValue,
 98     IN NDIS_PARAMETER_TYPE ParamType);
 99
100 STATIC VOID
101 WriteSingleParameter(
102     IN NDIS_HANDLE ParametersHandle,
103     IN PWCHAR ValueName,
104     IN ULONG ValueData,
105     IN NDIS_PARAMETER_TYPE ParamType);
106
107 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
108 //
109 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
110 //
111 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
112
113 #ifdef ALLOC_PRAGMA
114 #pragma alloc_text(INIT, ConfigureDriver)
115 #pragma alloc_text(INIT, ReadSingleParameter)
116 #pragma alloc_text(INIT, WriteSingleParameter)
117 #endif
118
119 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
120 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
121
122 #pragma NDIS_INIT_FUNCTION(DriverEntry)
123
124 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
125 //
126 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓
127 //
128 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
129 NTSTATUS
130 DriverEntry(
131     IN PDRIVER_OBJECT DriverObject,
132     IN PUNICODE_STRING RegistryPath)
133 {
134     NDIS_STATUS Status;
135     NDIS_PROTOCOL_CHARACTERISTICS ProtocolChars;
136     NDIS_MINIPORT_CHARACTERISTICS MiniportChars;
137     NDIS_STRING IMName = NDIS_STRING_CONST( "IM");
138     ULONG InitShutdownMask;
139     PWCHAR EventLogString = IMDriverName.Buffer;
140     PVOID DumpData;
141
142 #ifdef DBG
143     TNSMakeBeep();
144 #endif
145 .   D((0, "TNSEmul DriverEntry\n"));
146     D((0, "TNSEMUL, Built %s at %s\n", __DATE__, __TIME__));
147
148
149 .   IMDriverObject = DriverObject;
150
151
152     InitializeListHead( &AdapterList );
153     NdisAllocateSpinLock( &AdapterListLock );
154
155     NdisMInitializeWrapper( &HPWrapperHandle, DriverObject, RegistryPath, NULL );
156
157     InitShutdownMask = SHUTDOWN_TERMINATE_WRAPPER;
158
159     Status = ConfigureDriver( RegistryPath, &ConfigData ); ▓▓▓▓▓▓▓▓▓▓▓
160
161     if ( !NT_SUCCESS( Status )) {
162         D((0, "ConfigureDriver - Status: 0x%x\n", Status ));
163         goto DriverEntryError;
164     }
```

**File: D:\nt4DDK\src\tlmesn\tnsdrvr\tnsemul.c**                    **Page 3 of 20**

```
165
166
167      NdisZeroMemory(&ProtocolChars, sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
168      ProtocolChars.Name.Length = IMName.Length;
169      ProtocolChars.Name.Buffer = (PVOID)IMName.Buffer;
170
171      ProtocolChars.MajorNdisVersion = 4;
172      ProtocolChars.MinorNdisVersion = 0;
173
174      ProtocolChars.OpenAdapterCompleteHandler = LowerMPOpenAdapterComplete;
175      ProtocolChars.CloseAdapterCompleteHandler = LowerMPCloseAdapterComplete;
176      ProtocolChars.SendCompleteHandler = CLSendComplete;
177      ProtocolChars.TransferDataCompleteHandler = CLTransferDataComplete;
178      ProtocolChars.ResetCompleteHandler = CLResetComplete;
179      ProtocolChars.RequestCompleteHandler = CLRequestComplete;
180      ProtocolChars.ReceiveHandler = CLReceiveIndication;
181      ProtocolChars.ReceiveCompleteHandler = CLReceiveComplete;
182      ProtocolChars.StatusHandler = CLStatusIndication;
183      ProtocolChars.StatusCompleteHandler = CLStatusIndicationComplete;
184      //ProtocolChars.ReceivePacketHandler = CLReceivePacket;
185      ProtocolChars.ReceivePacketHandler = NULL;
186      ProtocolChars.BindAdapterHandler = BindToLowerMP;
187      ProtocolChars.UnbindAdapterHandler = UnbindFromLowerMP;
188      ProtocolChars.UnloadHandler = CLUnloadProtocol;
189
190      NdisRegisterProtocol(&Status,
191          &ClientProtocolHandle,
192          &ProtocolChars,
193          sizeof(NDIS_PROTOCOL_CHARACTERISTICS) + ProtocolChars.Name.Length);
194
195      if ( !NT_SUCCESS( Status )) {
196          D((0, "DoProtocolInit: couldn't register client handlers %08X\n", Status ));
197      }
198
199
200      if ( !NT_SUCCESS( Status )) {
201
202          D((0, "DoProtocolInit Failed! Status: 0x%x\n", Status));
203
204          DumpData = &Status;
205          NdisWriteErrorLogEntry(IMDriverObject,
206              EVENT_TRANSPORT_REGISTER_FAILED,
207              TNS_ERROR_PROTOCOL_INIT,
208              1,
209              &EventLogString,
210              sizeof( Status ),
211              DumpData);
212
213          goto DriverEntryError;
214      }
215
216      InitShutdownMask |= SHUTDOWN_DEREGISTER_PROTOCOL;
217
218      NdisZeroMemory(&MiniportChars, sizeof(NDIS_MINIPORT_CHARACTERISTICS));
219      MiniportChars.MajorNdisVersion = 4;
220      MiniportChars.MinorNdisVersion = 0;
221
222      MiniportChars.Reserved = 0;
223      MiniportChars.HaltHandler = MPHalt;
224      MiniportChars.InitializeHandler = MPInitialize;
225      MiniportChars.QueryInformationHandler = MPQueryInformation;
226      MiniportChars.ResetHandler = MPReset;
227      MiniportChars.SetInformationHandler = MPSetInformation;
228      MiniportChars.TransferDataHandler = MPTransferData;
229
230      MiniportChars.ReconfigureHandler = NULL;
231      MiniportChars.DisableInterruptHandler = NULL;
232      MiniportChars.EnableInterruptHandler = NULL;
233      MiniportChars.HandleInterruptHandler = NULL;
234      MiniportChars.ISRHandler = NULL;
235      MiniportChars.CheckForHangHandler = NULL;
236
237
238      MiniportChars.ReturnPacketHandler = MPReturnPacket;
239      MiniportChars.SendPacketsHandler = MPSendPackets;
240      MiniportChars.AllocateCompleteHandler = NULL;
241      MiniportChars.SendHandler = NULL;
242
243      Status = NdisIMRegisterLayeredMiniport(MPWrapperHandle,
244          &MiniportChars,
245          sizeof(MiniportChars),
246          &LMDriverHandle);
```

```
247
248      if ( !NT_SUCCESS( Status )) {
249
250          D((0, "DoMiniportInit Failed! Status: 0x%x\n", Status));
251
252          DumpData = &Status;
253          NdisWriteErrorLogEntry(IMDriverObject,
254              (ULONG)TNS_EVENT_MINIPORT_REGISTER_FAILED,
255              0,
256              1,
257              &EventLogString,
258              sizeof( Status ),
259              DumpData);
260
261          goto DriverEntryError;
262      }
263
264      Status = WDMInitialize( DriverObject, &InitShutdownMask );
265
266      if ( !NT_SUCCESS( Status )) {
267
268          D((0, "WDMInitialize Failed! Status: 0x%x\n", Status));
269
270          goto DriverEntryError;
271      }
272
273
274      return (STATUS_SUCCESS);
275
276
277 DriverEntryError:
278      //
279      if ( InitShutdownMask & SHUTDOWN_DEREGISTER_PROTOCOL ) {
280          if ( ClientProtocolHandle ) {
281              NdisDeregisterProtocol( &Status, ClientProtocolHandle );
282              if ( Status == NDIS_STATUS_PENDING ) {
283                  D((0, "Client DeregProto failed - %08X\n", Status));
284              }
285          }
286      }
287
288      if ( InitShutdownMask & SHUTDOWN_TERMINATE_WRAPPER ) {
289          NdisTerminateWrapper( MPWrapperHandle, NULL );
290      }
291
292      WDMCleanup( InitShutdownMask );
293
294      NdisFreeSpinLock( &AdapterListLock );
295      NdisFreeSpinLock( &PSAListLock );
296
297      return (STATUS_UNSUCCESSFUL);
298
299 } //DriverEntry
300
301 VOID
302 CLResetComplete(
303      IN  NDIS_HANDLE ProtocolBindingContext,
304      IN  NDIS_STATUS Status)
305 {
306      PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
307      D((0, "(%08X) CLResetComplete: Status = %08x\n", pAdapter, Status));
308 }
309
310 VOID
311 CLStatusIndication(
312      IN  NDIS_HANDLE ProtocolBindingContext,
313      IN  NDIS_STATUS GeneralStatus,
314      IN  PVOID       StatusBuffer,
315      IN  UINT        StatusBufferSize)
316 {
317      PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
318
319      D((0, "(%08X) CLStatusIndication: Status %08X\n", pAdapter, GeneralStatus));
320      //
321      //Now indicate the status to the upper layer
322      //
323      if (pAdapter->TNSDriverInitialized) {
324          NdisMIndicateStatus( pAdapter->TNSNdisHandle, GeneralStatus, StatusBuffer, StatusBufferSize )
325      }
326
327 } //CLStatusIndication
328
```

```
329  VOID
330  CLStatusIndicationComplete(
331      IN  NDIS_HANDLE ProtocolBindingContext)
332  {
333      PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
334      D((0, "(%08X) CLStatusIndicationComplete\n", pAdapter));
335
336      if (pAdapter->TNSDriverInitialized) {
337          NdisMIndicateStatusComplete(pAdapter->TNSNdisHandle);
338      }
339  } //CLStatusIndication
340
341
342
343  NTSTATUS
344  ConfigureDriver (
345      IN PUNICODE_STRING RegistryPath,
346      IN PCONFIG_DATA ConfigurationInfo)
347  {
348      NDIS_HANDLE ConfigHandle;
349      NDIS_STATUS Status;
350      NDIS_STRING TnsBlahBlah = NDIS_STRING_CONST("BlahBlah");
351      PNDIS_CONFIGURATION_PARAMETER pConfigParameter;
352
353      NdisOpenProtocolConfiguration( &Status, &ConfigHandle, RegistryPath );
354
355
356      ConfigurationInfo->PacketPoolSize = 200;
357
358      //
359      // Set parameters that are not dependent upon the size selected
360      //
361
362      ConfigurationInfo->DebugLevel = 10;
363      ConfigurationInfo->DebugMask = 0xffffffff;
364
365      if ( NT_SUCCESS( Status )) {
366
367          READ_HIDDEN_CONFIG ( PacketPoolSize, NdisParameterInteger );
368          NdisCloseConfiguration( ConfigHandle );
369      }
370
371      return STATUS_SUCCESS;
372  } //ConfigureDriver
373
374  STATIC ULONG
375  ReadSingleParameter(
376      IN HANDLE ConfigHandle,
377      IN PWCHAR ValueName,
378      IN ULONG DefaultValue,
379      IN NDIS_PARAMETER_TYPE NdisParamType)
380  {
381      UNICODE_STRING ValueKeyName;
382      ULONG ReturnValue;
383      NDIS_STATUS Status;
384      PNDIS_CONFIGURATION_PARAMETER ConfigParam;
385
386      MyAssert( NdisParamType == NdisParameterInteger || NdisParamType == NdisParameterHexInteger );
387
388      NdisInitUnicodeString( &ValueKeyName, ValueName );
389
390      NdisReadConfiguration(&Status,
391          &ConfigParam,
392          ConfigHandle,
393          &ValueKeyName,
394          NdisParamType);
395
396      if ( NT_SUCCESS( Status )) {
397          ReturnValue = ConfigParam->ParameterData.IntegerData;
398      } else {
399          ReturnValue = DefaultValue;
400      }
401
402      return ReturnValue;
403  } //ReadSingleParameter
404
405  VOID
406  BindToLowerMP(
407      OUT PNDIS_STATUS        Status,
408      IN  NDIS_HANDLE         BindContext,
409      IN  PNDIS_STRING        MPDeviceName,
410      IN  PVOID               SystemSpecific1,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**                    **Page 6 of 2**

```
411      IN  PVOID              SystemSpecific2)
412  {
413      PADAPTER pAdapter;
414      int i;
415      NDIS_STATUS OpenAdapterStatus;
416      NDIS_STATUS OpenErrorStatus;
417      NDIS_STATUS LocalStatus;
418      NDIS_MEDIUM MediumArray[] = {
419          NdisMediumFddi,
420          NdisMedium802_5,
421          NdisMedium802_3,
422          NdisMediumWan };
423
424      UINT MediumArraySize = sizeof( MediumArray ) / sizeof( NDIS_MEDIUM );
425      UINT MediaIndex;
426      ULONG AdapterStructSize;
427      ULONG NdisPacketTypes;
428      int j;
429
430      D((0, "BindToLowerMP: %s\n", MPDeviceName->Buffer ));
431
432      //
433      // Allocate enough space for the structure and two unicode buffers to hold
434      // the "IM " and underlying MP device names.  We add 3 extra Unicode char to
435      // to the IM device name to hold the "IM" addition to the MP name, that is be appended later
436      // on and another unicode char to separate the two strings for reading a
437      // The IM adapter will have the form "\Device\IM IEEPRO3" for example if it is sitting on top of
438      // IEEPRO3
439      //
440
441      AdapterStructSize = sizeof( ADAPTER ) + MPDeviceName->Length // space for the miniport device nam
442          + MPDeviceName->Length +                                // space for the IM virtual adapter
-2   3*sizeof(MPDeviceName);
443          4 * sizeof( UNICODE_NULL );
444
445      *Status = NdisAllocateMemory(&pAdapter, AdapterStructSize, 0, HighAddress);
446
447      if ( pAdapter == NULL ) {
448          PWCHAR StringData[2];
449
450          StringData[0] = IMDriverName.Buffer;
451          StringData[1] = L"Adapter";
452          NdisWriteErrorLogEntry(IMDriverObject,
453              (ULONG)EVENT_TRANSPORT_RESOURCE_POOL,
454              0,
455              2,
456              &StringData,
457              0,
458              NULL);
459
460          *Status = NDIS_STATUS_RESOURCES;
461          return;
462      }
463
464      NdisZeroMemory(pAdapter, AdapterStructSize);
465
466      GetProcessorSpeed(pAdapter);
467
468      //
469      // Get the ComputerName
470      //
471      {
472          HANDLE ParamHandle;
473          UNICODE_STRING KeyNameU;
474          HANDLE ConfigHandle;
475          ULONG Disposition;
476          OBJECT_ATTRIBUTES TmpObjectAttributes;
477          char nameBuf[256];
478          STRING ntNameString;
479          PKEY_VALUE_FULL_INFORMATION pKeyInfo;
480          unsigned char keyBuffer[128];
481          ULONG ResultLength;
482          unsigned short *pwString;
483          UNICODE_STRING ValueNameU;
484          NTSTATUS Status;
485
486          (VOID)sprintf(nameBuf, "\\Registry\\Machine\\System\\CurrentControlSet\\Control\\ComputerName
-2   tiveComputerName");
487          RtlInitString(&ntNameString, nameBuf);
488
489          Status = RtlAnsiStringToUnicodeString(
490              &KeyNameU,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**                      **Page 7 of 20**

```
491            &ntNameString,
492            TRUE);
493
494        if (Status == STATUS_SUCCESS) {
495
496            (VOID)sprintf(nameBuf, "ComputerName");
497            RtlInitString(&ntNameString, nameBuf);
498
499            Status = RtlAnsiStringToUnicodeString(
500                &ValueNameU,
501                &ntNameString,
502                TRUE);
503
504            InitializeObjectAttributes(
505                &TmpObjectAttributes,
506                &KeyNameU,
507                OBJ_CASE_INSENSITIVE,
508                NULL,
509                NULL);
510
511            Status = ZwCreateKey(
512                &ConfigHandle,
513                KEY_READ,
514                &TmpObjectAttributes,
515                0,
516                NULL,
517                0,
518                &Disposition);
519
520            Status = ZwQueryValueKey(
521                ConfigHandle,
522                &ValueNameU,
523                KeyValueFullInformation,
524                &keyBuffer,
525                sizeof(keyBuffer),
526                &ResultLength);
527
528            if (Status == STATUS_SUCCESS) {
529                int i;
530                pKeyInfo = (PKEY_VALUE_FULL_INFORMATION) keyBuffer;
531
532                ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
533                ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
534                ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
535
536                pwString = (unsigned short *)pKeyInfo;
537                ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
538                pwString = (unsigned short *)((ULONG)pwString + pKeyInfo->DataOffset);
539                ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
540
541                i=0;
542                while (*pwString && (i<MAX_COMPUTER_NAME_SIZE) ) {
543                    ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
544                    pAdapter->LocalComputerName[i++] = (unsigned char) *pwString;
545                    pwString++;
546                }
547
548                D((0, "Machine Name => %s\n", pAdapter->LocalComputerName));
549            }
550        }
551        ░░
552        ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
553        ░░
554        RtlFreeUnicodeString(&KeyNameU);
555        RtlFreeUnicodeString(&ValueNameU);
556    }
557
558    ░░
559    ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
560    ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
561    ░░
562    for (i=0; i<HARDWARE_ADDRESS_LENGTH; i++) {
563        pAdapter->SMNMacAddress[i] = 0xff;
564    }
565
566    ░░
567    ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
568    ░░
569    for (i=0; i<MAX_TEAM_NODES; i++) {
570        for (j=0; j<HARDWARE_ADDRESS_LENGTH; j++) {
571            pAdapter->TeamNodeTable[i].TNMacAddress[j] = 0x00;
572        }
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**          **Page 8 of 20**

```
573        pAdapter->TeamNodeTable[i].TNNodeID = 0xffffffff;
574    }
575
576    //
577    // Set adapter struct size so we know what size to use
578    // later when we free it
579    //
580    pAdapter->AdapterStructSize = AdapterStructSize;
581
582    //
583    // Init the structures we need to manage client and server
584    // request queues
585    //
586
587    InitializeListHead(&pAdapter->ClientWorkerListEntry);
588    InitializeListHead(&pAdapter->ServerWorkerListEntry);
589    InitializeListHead(&pAdapter->WorkerListEntryPool);
590
591    KeInitializeSemaphore(&pAdapter->ClientWorkerRequestSemaphore,
592        0,
593        MAXLONG);
594    KeInitializeSemaphore(&pAdapter->ClientWorkerResponseSemaphore,
595        0,
596        MAXLONG);
597    KeInitializeSemaphore(&pAdapter->ServerWorkerRequestSemaphore,
598        0,
599        MAXLONG);
600    KeInitializeSpinLock(&pAdapter->ClientWorkerListSpinLock);
601    KeInitializeSpinLock(&pAdapter->ServerWorkerListSpinLock);
602    KeInitializeSpinLock(&pAdapter->ListEntryPoolLock);
603
604    KeInitializeSpinLock(&pAdapter->MyStatsLock);
605
606    pAdapter->ListEntryItems = 50;
607
608    for (i=0; i<(int)pAdapter->ListEntryItems; i++) {
609        PREQUEST_DATA pRqstData;
610
611        pRqstData = (PREQUEST_DATA) ExAllocatePool(NonPagedPool, sizeof(REQUEST_DATA) );
612
613        if (pRqstData != NULL) {
614            ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
615                &pRqstData->Linkage,
616                &pAdapter->ListEntryPoolLock);
617
618        } else {
619            D((0, "Cannot allocate worker queue pool\n"));
620            _asm int 3
621        }
622    }
623
624
625    //
626    // Init the device name pointer to the buffer allocated at the back of the
627    // struct. The MPDevice name looks like \Device\EE163. This is the lower adapter that we want
628    // to bind to. We want to construct an interface device that looks like \Device\TN_EE163 and so
-2         adapter
629    // on until MY publishing
630    //
631
632    pAdapter->TNSDeviceName.MaximumLength = MPDeviceName->MaximumLength + 3 * sizeof( UNICODE_NULL );
-2    MPDeviceName
633    pAdapter->TNSDeviceName.Length = pAdapter->TNSDeviceName.MaximumLength;
634    pAdapter->TNSDeviceName.Buffer = (PWSTR)( pAdapter + 1 );
635
636    pAdapter->MPDeviceName.MaximumLength = MPDeviceName->Length;
637    pAdapter->MPDeviceName.Length = pAdapter->MPDeviceName.MaximumLength;
638    pAdapter->MPDeviceName.Buffer = (PWSTR)((PCHAR)pAdapter->TNSDeviceName.Buffer +
639        pAdapter->TNSDeviceName.MaximumLength +
640        sizeof( UNICODE_NULL ));
641
642    // Copy in our base device description
643    //
644    RtlCopyMemory(pAdapter->TNSDeviceName.Buffer, L"\\Device\\IM_", sizeof(L"\\Device\\IM_"));
645
646    // Now copy in the device name passed to us. This name is based on our instance in registry
647    // and looks like \Device\NNN and is preceded by the \IM word above
648
649    RtlCopyMemory(&(pAdapter->TNSDeviceName.Buffer[sizeof("\\Device\\IM")]),
650        &(MPDeviceName->Buffer[sizeof("\\Device")]),
651        MPDeviceName->Length - sizeof(L"\\Device"));
652
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**          **Page 9 of 20**

```
653
654
655     // LocalStatus = GetAdapterRegistryData((PNDIS_STRING)SystemSpecific1, Adapter);
656
657     if (0) {
658         D((0, "(%08X) BindToLowerMP: Couldn't get registry data %08X (%s)\n",
659             pAdapter, LocalStatus, MPDeviceName->Buffer ));
660
661         *Status = NDIS_STATUS_FAILURE;
662         NdisFreeMemory(pAdapter, (sizeof(ADAPTER)+MPDeviceName->Length+MPDeviceName->Length+4*sizeof(UNIC
-2 ODE_NULL)) , 0);
663         return;
664     }
665
666     //
667     // Init the event now since we use it in the completion handler
668     //
669     // Remember our binding context so we can complete BindAdapter later on
670     //
671     NdisInitializeEvent(&pAdapter->BlockingEvent);
672     // NdisInitializeEvent(&Adapter->SCLReceiveIndicationPacketCallBackEvent);
673     pAdapter->BindContext = BindContext;
674
675
676     // Open the adapter below us.
677     NdisOpenAdapter(&OpenAdapterStatus,
678         &OpenErrorStatus,
679         &(pAdapter->LowerMPHandle),
680         &MediaIndex,
681         MediumArray,
682         MediumArraySize,
683         ClientProtocolHandle,
684         pAdapter,
685         MPDeviceName,
686         0,
687         NULL);
688
689
690     if ( OpenAdapterStatus == NDIS_STATUS_PENDING ) {
691         NdisWaitEvent( &pAdapter->BlockingEvent, 0 );
692         NdisResetEvent( &pAdapter->BlockingEvent );
693     } else {
694         pAdapter->FinalStatus = OpenAdapterStatus;
695     }
696
697     if ( NT_SUCCESS( pAdapter->FinalStatus )) {
698
699         pAdapter->MediaType = MediumArray[ MediaIndex ];
700
701         if (pAdapter->MediaType = NdisMediumWan)
702             pAdapter->MediaType = NdisMedium802_3;
703     }
704     ProcessLowerMPOpenAdapter( pAdapter, pAdapter->FinalStatus );
705     pAdapter->TNSClientNodeID = 0xffffffff;
706
707     if (TNSSharedMemoryNodeEmulation == FALSE) {
708         if (PsCreateSystemThread(
709             &pAdapter->ClientWorkerThreadHandle,
710             (ACCESS_MASK) 0,
711             (POBJECT_ATTRIBUTES) NULL,
712             (HANDLE) NULL,
713             (PCLIENT_ID) NULL,
714             TNSClientWorkerThread,
715             (PVOID) pAdapter) != STATUS_SUCCESS) {
716
717             D((0, "Could not create client thread\n"));
718             _asm int 3
719         }
720     } else {
721         if (PsCreateSystemThread(
722             &pAdapter->ServerWorkerThreadHandle,
723             (ACCESS_MASK) 0,
724             (POBJECT_ATTRIBUTES) NULL,
725             (HANDLE) NULL,
726             (PCLIENT_ID) NULL,
727             TNSServerWorkerThread,
728             (PVOID) pAdapter) != STATUS_SUCCESS) {
729
730             D((0, "Could not Server worker thread\n"));
731             _asm int 3
732         }
733     }
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**         **Pag  1  of 2**

```
734
735      *Status = pAdapter->FinalStatus;
736
737  ) //ProcessLowerMP
738
739  STATIC NDIS_STATUS
740  GetAdapterRegistryData(
741      PNDIS_STRING IMParamsKey,
742      PADAPTER pAdapter)
743  {
744      NDIS_STATUS Status;
745      NDIS_HANDLE ConfigHandle;
746      NDIS_STRING IMInstanceNumberKey = NDIS_STRING_CONST( "InstanceNumber" );
747      PNDIS_CONFIGURATION_PARAMETER ConfigParam;
748
749      NdisOpenProtocolConfiguration( &Status, &ConfigHandle, IMParamsKey );
750
751      if ( !NT_SUCCESS( Status )) {
752          D((0, "(%08X) GetAdapterRegistryData: can't open key %s (%08X)\n", pAdapter, IMParamsKey->Buffer,
-2   Status ));
753          BreakPoint();
754          return Status;
755      }
756
757      //
758      //get the IM device instance number and build the device instance string
759      //
760      NdisReadConfiguration(&Status,
761          &ConfigParam,
762          ConfigHandle,
763          &IMInstanceNumberKey,
764          NdisParameterInteger);
765
766
767      if ( !NT_SUCCESS( Status )) {
768          D((0, "(%08X) GetAdapterRegistryData: Missing InstanceNumber key\n", pAdapter));
769
770          Status = NDIS_STATUS_FAILURE;
771          goto CloseConfig;
772      }
773
774      pAdapter->DevInstance = (USHORT)ConfigParam->ParameterData.IntegerData;
775
776      NdisMoveMemory(pAdapter->TNSDeviceName.Buffer, IMMPName.Buffer, IMMPName.Length);
777
778      pAdapter->TNSDeviceName.Buffer[ IMMPName.Length / sizeof( WCHAR ) ] = L'0' + pAdapter->DevInstance;
779
780
781  CloseConfig:
782      NdisCloseConfiguration( ConfigHandle );
783
784      return Status;
785
786  ) //GetAdapterRegistryData
787
788  STATIC VOID
789  ProcessLowerMPOpenAdapter(
790      IN  PADAPTER pAdapter,
791      IN  NDIS_STATUS Status)
792  {
793      NTSTATUS EventStatus;
794      NDIS_HARDWARE_STATUS HWStatus;
795      NDIS_MEDIA_STATE MediaState = 0xFFFFFFFF;
796      NDIS_STRING IMDevName;
797      ULONG MacOptions;
798      ULONG ErrorLogData[2];
799      PWCHAR StringData[2];
800      PVOID DumpData;
801
802      D((0, "(%08X) ProcessLowerMPOpenAdapter\n", pAdapter));
803      //
804      //process the lower MP Clear and get input
805      //
806
807      if ( !NT_SUCCESS( Status )) {
808          D((0, "(%08X) ProcessLowerMPOpenAdapter: binding failed %08X\n", pAdapter, Status));
809          if ( Status == NDIS_STATUS_ADAPTER_NOT_FOUND ) {
810              EventStatus = EVENT_TRANSPORT_ADAPTER_NOT_FOUND;
811          } else {
812              EventStatus = EVENT_TRANSPORT_BINDING_FAILED;
813          }
814
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**          **Page 11 of 2**

```
815        StringData[0] = pAdapter->TNSDeviceName.Buffer;
816        StringData[1] = pAdapter->MPDeviceName.Buffer;
817        DumpData = &Status;
818
819        NdisWriteErrorLogEntry(IMDriverObject,
820            EventStatus,
821            0,
822            2,
823            &StringData,
824            sizeof( Status ),
825            DumpData);
826
827        NdisFreeMemory(pAdapter, pAdapter->AdapterStructSize, 0);
828        return;
829    }
830
831    D((0, "(%08X) =1 Adapter\n", pAdapter ));
832    InitializeListHead( &pAdapter->ClientList );
833    pAdapter->ShutdownMask = 0;
834
835
836    NdisInterlockedInsertTailList(&AdapterList, &pAdapter->Linkage, &AdapterListLock);
837        {
838    Status = MakeLocalNdisRequest(pAdapter,
839        OID_GEN_HARDWARE_STATUS,
840        &HWStatus,
841        sizeof(HWStatus));
842
843    if ( Status == NDIS_STATUS_INVALID_OID || HWStatus == NdisHardwareStatusReady ) {
844
845        Status = MakeLocalNdisRequest(pAdapter,
846            OID_GEN_MEDIA_CONNECT_STATUS,
847            &MediaState,
848            sizeof( MediaState ));
849
850        if ( Status == NDIS_STATUS_INVALID_OID || MediaState == NdisMediaStateConnected ) {
851
852            Status = MakeLocalNdisRequest(pAdapter,
853                OID_GEN_LINK_SPEED,
854                &pAdapter->LinkSpeed,
855                sizeof( pAdapter->LinkSpeed ));
856
857            if ( !NT_SUCCESS( Status )) {
858
859                D((0, "(%08X) ProcessLowerMPOpenAdapter: Can't get link speed - Status %08X\n", pAdapter,
-2  Status));
860
861                ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
862                ErrorLogData[ 1 ] = OID_GEN_LINK_SPEED;
863
864                NdisWriteErrorLogEntry(pAdapter->LowerMPHandle,
865                    NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER,
866                    2,
867                    ErrorLogData);
868
869                return;
870            }
871
872        } else {
873
874            D((0, "(%08X) ProcessLowerMPOpenAdapter: Media not connected\n", pAdapter ));
875        }
876    } else {
877
878        D((0, "(%08X) ProcessLowerMPOpenAdapter: HW Status not ready (%d)\n", HWStatus));
879    }
880
881    Status = MakeLocalNdisRequest(
882        pAdapter,
883        OID_802_3_CURRENT_ADDRESS,
884        &pAdapter->LowerMPMacAddress,
885        HARDWARE_ADDRESS_LENGTH);
886
887    if ( NT_SUCCESS( Status )) {
888        D((0, "ProcessLowerMPOpenAdapter: got hardware address -> %02x %02x %02x %02x %02x %02x \n",
889            pAdapter->LowerMPMacAddress[0],
890            pAdapter->LowerMPMacAddress[1],
891            pAdapter->LowerMPMacAddress[2],
892            pAdapter->LowerMPMacAddress[3],
893            pAdapter->LowerMPMacAddress[4],
894            pAdapter->LowerMPMacAddress[5]));
895    } else {
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**                    **Page 12 of 2**

```
896        D((0, "ProcessLowerMPOpenAdapter: can't get hardware address \n" ));
897    }
898
899    Status = MakeLocalNdisRequest(pAdapter,
900        OID_GEN_MAC_OPTIONS,
901        &MacOptions,
902        sizeof(MacOptions));
903
904    if ( NT_SUCCESS( Status )) {
905        pAdapter->CopyLookaheadData = (BOOLEAN)(MacOptions & NDIS_MAC_OPTION_COPY_LOOKAHEAD_DATA);
906    }
907
908    Status = AllocatePacketPool(pAdapter);
909
910    if (!NT_SUCCESS(Status)) {
911        return;
912    }
913
914    Status = AllocateReceiveBufferPools(pAdapter);
915
916    if (!NT_SUCCESS(Status)) {
917        return;
918    }
919
920    NdisInitUnicodeString( &IMDevName, &pAdapter->TNSDeviceName.Buffer[8] );
921
922
923    CurrentAdapter = pAdapter;  .
924
925    D((0, "Calling NdisIMinitializeDeviceInstance\n"));
926    Status = NdisIMInitializeDeviceInstance(LMDriverHandle, &IMDevName);
927
928    if ( !NT_SUCCESS( Status )) {
929
930        D((0, "(%08X) ProcessLowerMPOpenAdapter: can't init IM device %s (%08X)\n",
931            pAdapter, IMDevName.Buffer, Status));
932
933        ErrorLogData[ 0 ] = TNS_ERROR_CANT_INITIALIZE_IMSAMP_DEVICE;
934        ErrorLogData[ 1 ] = Status;
935
936        NdisWriteErrorLogEntry(pAdapter->LowerMPHandle,
937            NDIS_ERROR_CODE_DRIVER_FAILURE,
938            2,
939            ErrorLogData);
940
941        return;
942    }
943
944    pAdapter->ShutdownMask |= SHUTDOWN_DEINIT_DEV_INSTANCE;
945
946    return;
947
948 }  ██████████████████████████
949
950 VOID
951 LowerMPOpenAdapterComplete(
952     IN  PADAPTER pAdapter,
953     IN  NDIS_STATUS Status,
954     IN  NDIS_STATUS OpenErrorStatus)
955 {
956     NDIS_MEDIA_STATE MediaState = 0xFFFFFFFF;
957
958     D((0, "(%08X) LowerMPOpenAdapterComplete\n", pAdapter));
959
960     pAdapter->FinalStatus = Status;
961     NdisSetEvent( &pAdapter->BlockingEvent );
962
963 }  ██████████████████████████
964
965 STATIC NDIS_STATUS
966 AllocatePacketPool(
967     PADAPTER pAdapter)
968 {
969     NDIS_STATUS Status;
970     ULONG ProtoReservedSize;
971
972     ████████████████████████████████████████████████████████████
973     ██
974
975     ProtoReservedSize = sizeof(TNS_PACKET_CONTEXT);
976
977     NdisAllocatePacketPool(&Status,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**          **Page 13 of 20**

```
978              &pAdapter->PacketPoolHandle,
979              ConfigData.PacketPoolSize,
980              ProtoReservedSize);
981
982      return Status;
983
984  }    // AllocatePacketPool
985
986  STATIC NDIS_STATUS
987  AllocateReceiveBufferPools(
988      PADAPTER pAdapter)
989  {
990      NDIS_STATUS Status;
991      ULONG HeaderSize;
992      ULONG FrameSize;        // Doesn't include the header
993      NDIS_ERROR_CODE ErrorCode;
994      ULONG ErrorLogData[2];
995
996      //
997      // max amount of data w/o the MAC header
998      //
999      Status = MakeLocalNdisRequest(pAdapter,
1000             OID_GEN_MAXIMUM_FRAME_SIZE,
1001             &FrameSize,
1002             sizeof(FrameSize));
1003
1004      if ( !NT_SUCCESS( Status )) {
1005
1006          D((0, "(%08X) AllocateReceiveBufferPool: Can't get frame size - Status %08X\n", pAdapter, Status)
     -2  );
1007
1008          ErrorCode = NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER;
1009          ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
1010          ErrorLogData[ 1 ] = OID_GEN_MAXIMUM_FRAME_SIZE;
1011          goto ErrorExit;
1012      }
1013
1014      //
1015      // max frame including the header
1016      //
1017
1018      Status = MakeLocalNdisRequest(pAdapter,
1019             OID_GEN_MAXIMUM_TOTAL_SIZE,
1020             &pAdapter->TotalSize,
1021             sizeof(pAdapter->TotalSize));
1022
1023      if ( !NT_SUCCESS( Status )) {
1024
1025          D((0, "(%08X) AllocateReceiveBufferPool: Can't get total size - Status %08X\n", pAdapter, Status)
     -2  );
1026
1027          ErrorCode = NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER;
1028          ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
1029          ErrorLogData[ 1 ] = OID_GEN_MAXIMUM_TOTAL_SIZE;
1030
1031          goto ErrorExit;
1032      }
1033
1034      //
1035      // compute the actual header size
1036      //
1037      HeaderSize = pAdapter->TotalSize - FrameSize;
1038      D((0, "FrameSize => %d, HeaderSize => %d, TotalSize => %d\n", FrameSize, HeaderSize, pAdapter->TotalS
     -2  ize));
1039
1040      Status = MakeLocalNdisRequest(pAdapter,
1041             OID_GEN_MAXIMUM_LOOKAHEAD,
1042             &pAdapter->LookaheadBufferSize,
1043             sizeof(pAdapter->LookaheadBufferSize));
1044
1045      if ( !NT_SUCCESS( Status )) {
1046
1047          D((0, "(%08X) AllocateReceiveBufferPool: Can't get lookahead size - Status %08X\n", pAdapter, Sta
     -2  tus));
1048
1049          ErrorCode = NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER;
1050          ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
1051          ErrorLogData[ 1 ] = OID_GEN_MAXIMUM_LOOKAHEAD;
1052          goto ErrorExit;
1053      }
1054
1055      pAdapter->LookaheadBufferSize += HeaderSize;
```

```
1056
1057     //
1058     //Init the lookahead buffer pool
1059     //
1060     NdisAllocateBufferPool(&Status, &pAdapter->LookaheadPoolHandle, ConfigData.PacketPoolSize);
1061
1062     return Status;
1063
1064 ErrorExit:
1065
1066     NdisWriteErrorLogEntry(
1067         pAdapter->LowerMPHandle,
1068         ErrorCode,
1069         2,
1070         ErrorLogData );
1071
1072     return Status;
1073
1074 )    //AllocateReceiveBufferPool
1075
1076
1077 NDIS_STATUS
1078 MPInitialize(
1079     OUT PNDIS_STATUS        OpenErrorStatus,
1080     OUT PUINT               SelectedMediumIndex,
1081     IN  PNDIS_MEDIUM        MediumArray,
1082     IN  UINT                MediumArraySize,
1083     IN  NDIS_HANDLE         MiniportAdapterHandle,
1084     IN  NDIS_HANDLE         WrapperConfigurationContext)
1085 (
1086     NDIS_STRING LowerAdapterKey = NDIS_STRING_CONST( "LowerAdapter" );
1087     PADAPTER pAdapterInList;
1088     ULONG ErrorLogData[2];
1089     PNDIS_MINIPORT_BLOCK Mp = (PNDIS_MINIPORT_BLOCK)MiniportAdapterHandle;
1090     NDIS_STATUS Status;
1091     NDIS_HANDLE ConfigHandle;
1092     PNDIS_CONFIGURATION_PARAMETER pConfigParameter;
1093     NDIS_STRING TnsSmnModeString = NDIS_STRING_CONST("TNSSMNEmulationMode");
1094
1095
1096     D((0, "MPInitialize: Enter\n"));
1097     D((0, "MiniportInitialize Miniport->BaseName = %ws\n",Mp->MiniportName.Buffer ));
1098
1099     pAdapterInList = FindAdapterByName(Mp->MiniportName.Buffer);
1100
1101
1102     NdisOpenConfiguration(
1103         &Status,
1104         &ConfigHandle,
1105         WrapperConfigurationContext);
1106
1107     if (Status != STATUS_SUCCESS) (
1108         D((0, "Cannot open miniport config data\n"));
1109     ) else (
1110         NdisReadConfiguration(
1111             &Status,
1112             &pConfigParameter,
1113             ConfigHandle,
1114             &TnsSmnModeString,
1115             NdisParameterHexInteger);
1116
1117         if (Status != STATUS_SUCCESS) (
1118             D((0, "Can't read reg, Status => %x\n", Status));
1119         ) else (
1120             D((0, "read reg, value => %x\n", pConfigParameter->ParameterData.IntegerData));
1121             TNSSharedMemoryNodeEmulation = pConfigParameter->ParameterData.IntegerData;
1122         )
1123
1124     )
1125
1126     //Config value is in TNSSharedMemoryNodeEmulation
1127     //
1128     if ( !pAdapterInList ) (
1129
1130         D((0, "Can't find adapter for MP dev # %ws\n",Mp->MiniportName.Buffer));
1131
1132         ErrorLogData[ 0 ] = TNS_ERROR_BAD_REGISTRY_DATA;
1133         ErrorLogData[ 1 ] = TNS_ERROR_INVALID_IMSAMP_MP_INSTANCE;
1134
1135         NdisWriteErrorLogEntry(MiniportAdapterHandle,
1136             NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER,
1137             2,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**                    **Page 15 of 20**

```
1138          ErrorLogData);
1139
1140      BreakPoint();
1141      return NDIS_STATUS_FAILURE;
1142  }
1143
1144  //
1145  //  lookup our media type in the supplied media array
1146  //
1147  for (--MediumArraySize ; MediumArraySize > 0; ) {
1148      if ( MediumArray[ MediumArraySize ] == pAdapterInList->MediaType ) {
1149          break;
1150      }
1151      if ( MediumArraySize == 0 ) {
1152          break;
1153      }
1154      --MediumArraySize;
1155  }
1156
1157  if ( MediumArraySize == 0 && MediumArray[ 0 ] != pAdapterInList->MediaType ) {
1158      BreakPoint();
1159      return NDIS_STATUS_UNSUPPORTED_MEDIA;
1160  }
1161
1162  *SelectedMediumIndex = MediumArraySize;
1163
1164  //
1165  //  save the NDIS handle to this
1166  //
1167  pAdapterInList->TNSNdisHandle = MiniportAdapterHandle;
1168
1169  DM((DEBUG_INFO, DEBUG_MASKEN_INIT, "AdapterInList->TNSNdisHandle => %x\n", pAdapterInList->TNSNdisHan
-2 dle));
1170  //
1171  //  finish the initialization process by setting attributes
1172  //
1173  NdisMSetAttributesEx(MiniportAdapterHandle,
1174      pAdapterInList,
1175      0,
1176      NDIS_ATTRIBUTE_DESERIALIZE |
1177      NDIS_ATTRIBUTE_IGNORE_PACKET_TIMEOUT |
1178      NDIS_ATTRIBUTE_IGNORE_REQUEST_TIMEOUT |
1179      NDIS_ATTRIBUTE_INTERMEDIATE_DRIVER ,
1180      0);
1181
1182  //
1183  //  mark this adapter has been initialized
1184  //
1185  pAdapterInList->TNSDriverInitialized = TRUE;
1186
1187  return NDIS_STATUS_SUCCESS;
1188
1189  }  // MPInitialize
1190
1191  PADAPTER
1192  FindAdapterByName(
1193      PWCHAR AdapterName)
1194  {
1195      PLIST_ENTRY NextAdapter;
1196      PADAPTER pAdapterInList;
1197      ULONG NameLength = 0;
1198      PWCHAR pw = AdapterName;
1199
1200      while ( *pw++ != 0  && NameLength < 64 ) {
1201          ++NameLength;
1202      }
1203
1204      NameLength *= sizeof( WCHAR );
1205
1206      NdisAcquireSpinLock( &AdapterListLock );
1207
1208      NextAdapter = AdapterList.Flink;
1209      while ( NextAdapter != &AdapterList ) {
1210
1211          pAdapterInList = CONTAINING_RECORD( NextAdapter, ADAPTER, Linkage );
1212
1213          //  the name length includes the trailing NULL and our length doesn't
1214
1215          if ( pAdapterInList->TNSDeviceName.Length == (NameLength+2) ) {
1216              if ( NdisEqualMemory(pAdapterInList->TNSDeviceName.Buffer, AdapterName, NameLength)) {
1217                  break;
1218              }
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**                    **Page 16 of 20**

```
1219          )
1220
1221          NextAdapter = NextAdapter->Flink;
1222     )
1223
1224     if ( NextAdapter != &AdapterList ) {
1225     } else {
1226          pAdapterInList = NULL;
1227     )
1228
1229     NdisReleaseSpinLock( &AdapterListLock );
1230
1231     return pAdapterInList;
1232 )
1233
1234 VOID
1235 UnbindFromLowerMP(
1236     OUT PNDIS_STATUS          Status,
1237     IN  NDIS_HANDLE           ProtocolBindingContext,
1238     IN  NDIS_HANDLE           UnbindContext)
1239 {
1240     PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
1241     NDIS_STATUS LocalStatus;
1242
1243     D((0, "(%08X) UnbindFromLowerMP\n", pAdapter));
1244
1245     if ( pAdapter->ShutdownMask & SHUTDOWN_DEINIT_DEV_INSTANCE ) {
1246
1247          LocalStatus = NdisIMDeInitializeDeviceInstance(pAdapter->TNSNdisHandle);
1248          MyAssert(NT_SUCCESS (LocalStatus));
1249
1250          pAdapter->ShutdownMask &= ~SHUTDOWN_DEINIT_DEV_INSTANCE;
1251     )
1252
1253     pAdapter->BindContext = UnbindContext;
1254
1255     *Status = NDIS_STATUS_PENDING;
1256
1257 ) ▓▓▓▓▓▓▓▓▓▓▓▓
1258
1259 VOID
1260 LowerMPCloseAdapterComplete(
1261     IN  NDIS_HANDLE ProtocolBindingContext,
1262     IN  NDIS_STATUS Status)
1263 {
1264     PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
1265
1266     D((0, "(%08X) LowerMPCloseAdapterComplete\n", pAdapter));
1267
1268     MyAssert( NT_SUCCESS( Status ));
1269
1270     if ( pAdapter->BindContext ) {
1271          NdisCompleteUnbindAdapter( pAdapter->BindContext, Status );
1272     )
1273
1274     NdisAcquireSpinLock( &AdapterListLock );
1275     RemoveEntryList( &pAdapter->Linkage );
1276     NdisReleaseSpinLock( &AdapterListLock );
1277
1278     if ( pAdapter->ShutdownMask & SHUTDOWN_DEALLOC_PACKET_POOL ) {
1279
1280          NdisFreePacketPool( pAdapter->PacketPoolHandle );
1281     )
1282
1283     ▓▓
1284     ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
1285     ▓▓
1286
1287     if ( pAdapter->ShutdownMask & SHUTDOWN_DEALLOC_LOOKAHEAD_POOL ) {
1288
1289          NdisFreeBufferPool( pAdapter->LookaheadPoolHandle );
1290     )
1291
1292
1293     NdisFreeSpinLock( &pAdapter->Lock );
1294
1295     NdisFreeMemory(pAdapter, pAdapter->AdapterStructSize, 0);
1296
1297 ) ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
1298
1299 VOID
1300 CLUnloadProtocol(
```

```
1301      VOID)
1302  {
1303      BreakPoint();
1304  } //CLUnloadProtocol

1305
1306
1307  VOID
1308  MPHalt(
1309      IN  NDIS_HANDLE      MiniportAdapterContext)
1310  {
1311      PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
1312
1313      D((0, "(%08X) MPHalt\n", pAdapter));
1314      pAdapter->ShutdownMask &= ~SHUTDOWN_DEINIT_DEV_INSTANCE;
1315      BreakPoint();
1316  } //MPHalt
1317
1318  NDIS_STATUS
1319  MPReset(
1320      OUT PBOOLEAN        AddressingReset,
1321      IN  NDIS_HANDLE      MiniportAdapterContext)
1322  {
1323      PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
1324      D((0, "(%08X) MPReset\n", pAdapter));
1325      *AddressingReset = FALSE;
1326      return NDIS_STATUS_SUCCESS;
1327  } //MPReset
1328
1329  //****************************************************************
1330  //
1331  //Functions to handle making requests to the underlying miniport drivers,
1332  //and for us to gather information on the underlying miniport drivers.
1333  //This code is also in place to filter certain requests to/from the
1334  //underlying miniports.
1335  //
1336  //****************************************************************
1337
1338  NDIS_STATUS
1339  MakeLocalNdisRequest(
1340      PADAPTER pAdapter,
1341      NDIS_OID Oid,
1342      PVOID Buffer,
1343      ULONG BufferSize)
1344  {
1345      NDIS_STATUS Status;
1346      ULONG BytesNeeded, BytesWritten;
1347
1348      pAdapter->Request.RequestType = NdisRequestQueryInformation;
1349      pAdapter->Request.DATA.QUERY_INFORMATION.Oid = Oid;
1350      pAdapter->Request.DATA.QUERY_INFORMATION.InformationBuffer = Buffer;
1351      pAdapter->Request.DATA.QUERY_INFORMATION.InformationBufferLength = BufferSize;
1352      pAdapter->BytesNeeded = &BytesNeeded;
1353      pAdapter->BytesReadOrWritten = &BytesWritten;
1354      pAdapter->LocalRequest = TRUE;
1355
1356      NdisResetEvent( &pAdapter->BlockingEvent );
1357
1358      NdisRequest(&Status, pAdapter->LowerMPHandle, &pAdapter->Request);
1359
1360      //
1361      //Only wait if the MP pended our request
1362      //
1363      if (Status == NDIS_STATUS_PENDING) {
1364
1365          NdisWaitEvent( &pAdapter->BlockingEvent, 0 );
1366          NdisResetEvent( &pAdapter->BlockingEvent );
1367          Status = pAdapter->FinalStatus;
1368      }
1369
1370      //
1371      //Some MPs return an NT-specific status code. Ouch.
1372      //
1373      if ( Status == STATUS_NOT_SUPPORTED ) {
1374          Status = NDIS_STATUS_INVALID_OID;
1375      }
1376
1377      return Status;
1378  } //MakeLocalNdisRequest
1379
1380
1381
1382
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c                    Page 18 of 2

```
1383 NDIS_STATUS
1384 MakeLocalNdisRequestSet(
1385      PADAPTER pAdapter,
1386      NDIS_OID Oid,
1387      PVOID Buffer,
1388      ULONG BufferSize)
1389 {
1390      NDIS_STATUS Status;
1391      ULONG BytesNeeded, BytesWritten;
1392
1393      pAdapter->Request.RequestType = NdisRequestSetInformation;
1394      pAdapter->Request.DATA.QUERY_INFORMATION.Oid = Oid;
1395      pAdapter->Request.DATA.QUERY_INFORMATION.InformationBuffer = Buffer;
1396      pAdapter->Request.DATA.QUERY_INFORMATION.InformationBufferLength = BufferSize;
1397      pAdapter->BytesNeeded = &BytesNeeded;
1398      pAdapter->BytesReadOrWritten = &BytesWritten;
1399      pAdapter->LocalRequest = TRUE;
1400
1401      NdisResetEvent( &pAdapter->BlockingEvent );
1402
1403      NdisRequest(&Status, pAdapter->LowerMPHandle, &pAdapter->Request);
1404
1405      //
1406      // only wait if the MP pended our request
1407      //
1408      if (Status == NDIS_STATUS_PENDING) {
1409
1410          NdisWaitEvent( &pAdapter->BlockingEvent, 0 );
1411          NdisResetEvent( &pAdapter->BlockingEvent );
1412          Status = pAdapter->FinalStatus;
1413      }
1414
1415      if ( Status == STATUS_NOT_SUPPORTED ) {
1416          Status = NDIS_STATUS_INVALID_OID;
1417      }
1418
1419      D((0, "MakeLocalNdisRequestSet Status => %x\n", Status));
1420      return Status;
1421 } // MakeLocalNdisRequest
1422
1423
1424 NDIS_STATUS
1425 MPSetInformation(
1426      IN   NDIS_HANDLE     MiniportAdapterContext,
1427      IN   NDIS_OID        Oid,
1428      IN   PVOID           InformationBuffer,
1429      IN   ULONG           InformationBufferLength,
1430      OUT  PULONG          BytesRead,
1431      OUT  PULONG          BytesNeeded)
1432 {
1433      PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
1434      NDIS_STATUS Status;
1435      ULONG FoundFlag;
1436
1437      Status = NDIS_STATUS_FAILURE;
1438
1439      D((0, "MPSetInformation, Context => %x, (%x) NDIS_OID => %s\n", pAdapter, Oid, GetNDISOidString(Oid,
 -2 &FoundFlag) ));
1440
1441      // set up the Request and return the result
1442      //
1443      pAdapter->Request.RequestType = NdisRequestSetInformation;
1444      pAdapter->Request.DATA.SET_INFORMATION.Oid = Oid;
1445      pAdapter->Request.DATA.SET_INFORMATION.InformationBuffer = InformationBuffer;
1446      pAdapter->Request.DATA.SET_INFORMATION.InformationBufferLength = InformationBufferLength;
1447      pAdapter->BytesNeeded = BytesNeeded;
1448      pAdapter->BytesReadOrWritten = BytesRead;
1449
1450      NdisRequest(&Status, pAdapter->LowerMPHandle, &pAdapter->Request);
1451
1452      if (Status == NDIS_STATUS_SUCCESS) {
1453          *BytesRead = pAdapter->Request.DATA.SET_INFORMATION.BytesRead;
1454          *BytesNeeded = pAdapter->Request.DATA.SET_INFORMATION.BytesNeeded;
1455      }
1456
1457      return (Status);
1458 } // MPSetInformation
1459
1460
1461 NDIS_STATUS
1462 MPQueryInformation(
1463      IN   NDIS_HANDLE    MiniportAdapterContext,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**

```
1464      IN  NDIS_OID        Oid,
1465      IN  PVOID          InformationBuffer,
1466      IN  ULONG          InformationBufferLength,
1467      OUT PULONG         BytesWritten,
1468      OUT PULONG         BytesNeeded)
1469 {
1470      PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
1471      NDIS_STATUS Status = NDIS_STATUS_FAILURE;
1472      ULONG FoundFlag;
1473
1474      D((0, "MPQueryInformation, Context => %x, (%x) NDIS_OID -> %s\n", pAdapter, Oid, GetNDISOidString(Oid
     -2 , &FoundFlag) ));
1475
1476      pAdapter->Request.RequestType = NdisRequestQueryInformation;
1477      pAdapter->Request.DATA.QUERY_INFORMATION.Oid = Oid;
1478      pAdapter->Request.DATA.QUERY_INFORMATION.InformationBuffer = InformationBuffer;
1479      pAdapter->Request.DATA.QUERY_INFORMATION.InformationBufferLength = InformationBufferLength;
1480      pAdapter->BytesNeeded = BytesNeeded;
1481      pAdapter->BytesReadOrWritten = BytesWritten;
1482
1483      //
1484      // Default case, most requests will be passed to the miniport below
1485      //
1486      NdisRequest(&Status, pAdapter->LowerMPHandle ,&pAdapter->Request);
1487
1488      //
1489      // If the query was a success, pass the results back to the entity that made the request
1490      //
1491      if (Status == NDIS_STATUS_SUCCESS) {
1492          *BytesWritten = pAdapter->Request.DATA.QUERY_INFORMATION.BytesWritten;
1493          *BytesNeeded = pAdapter->Request.DATA.QUERY_INFORMATION.BytesNeeded;
1494      }
1495
1496      return(Status);
1497
1498 } // MPQueryInformation
1499
1500 VOID
1501 CLRequestComplete(
1502      IN  NDIS_HANDLE      ProtocolBindingContext,
1503      IN  PNDIS_REQUEST    NdisRequest,
1504      IN  NDIS_STATUS      Status)
1505 {
1506      PADAPTER pAdapter = (PADAPTER) ProtocolBindingContext;
1507      NDIS_OID Oid = pAdapter->Request.DATA.SET_INFORMATION.Oid;
1508      ULONG FoundFlag;
1509
1510      //
1511      // Command the set or query and copy info to the buffer NDIS_STATUS type(PADAPTER)BytesNeed.be.
1512      //
1513      if (pAdapter->LocalRequest) {
1514          pAdapter->LocalRequest = FALSE;
1515          NdisSetEvent(&pAdapter->BlockingEvent);
1516      } else {
1517          switch(NdisRequest->RequestType) {
1518              case NdisRequestQueryInformation:
1519
1520                  *pAdapter->BytesReadOrWritten = NdisRequest->DATA.QUERY_INFORMATION.BytesWritten;
1521
1522                  *pAdapter->BytesNeeded = NdisRequest->DATA.QUERY_INFORMATION.BytesNeeded;
1523
1524                  D((0, "CLRequest Complete, TNSNdisHandle => %x, Status => %x, (%x) Oid => %s\n",
1525                      pAdapter->TNSNdisHandle,
1526                      Status,
1527                      Oid,
1528                      GetNDISOidString(Oid, &FoundFlag)));
1529
1530                  NdisMQueryInformationComplete(pAdapter->TNSNdisHandle, Status);
1531
1532                  break;
1533
1534              case NdisRequestSetInformation:
1535
1536                  *pAdapter->BytesReadOrWritten = NdisRequest->DATA.SET_INFORMATION.BytesRead;
1537                  *pAdapter->BytesNeeded = NdisRequest->DATA.SET_INFORMATION.BytesNeeded;
1538
1539                  NdisMSetInformationComplete(pAdapter->TNSNdisHandle, Status);
1540                  break;
1541
1542              default:
1543                  ASSERT(0);
1544                  break;
```

```
1545          }
1546      }
1547  }  77=CuRequestComplete
1548
1549
1550
1551
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c**                    **Page 1 of 12**

```
 1  //**********************************************************************
 2  //
 3  // COPYRIGHT:
 4  //     This program is an unpublished work fully protected by the United
 5  //     States copyright laws and is considered a trade secret belonging to
 6  //     Times N Systems, Inc.  To the extent that this work may be
 7  //     considered "published," the following notice applies: (c) 1999, Times N
 8  //     Systems, Inc.  Any unauthorized use, reproduction, distribution,
 9  //     display, modification, or disclosure of this program is strictly
10  //     prohibited.
11  //
12  //**********************************************************************
13  //
14  //**********************************************************************
15  // Module:
16  //     recv.c - Times N intermediate driver to emulate high-speed
17  //             interconnect
18  //
19  // Description:
20  //     Routines to handle receiving data, and parsing Times N specific
21  //     interconnect messages.
22  //
23  // Environment:
24  //     Windows NT Kernel Mode, Ndis Driver models
25  //
26  // Exports:
27  //     See Module functions generated by script processing.
28  //
29  // Author:
30  //     Vince Bridgers
31  //     vinceb@timesn.com
32  //
33  //**********************************************************************
34
35  #include "tns.h"
36  #include "tnsdebug.h"
37  #include "x86.h"
38
39  VOID
40  MPReturnPacket(
41      IN  NDIS_HANDLE         MiniportAdapterContext,
42      IN  PNDIS_PACKET        Packet);
43
44  NDIS_STATUS
45  CLReceiveIndication(
46      IN  NDIS_HANDLE         ProtocolBindingContext,
47      IN  NDIS_HANDLE         MacReceiveContext,
48      IN  PVOID               HeaderBuffer,
49      IN  UINT                HeaderBufferSize,
50      IN  PVOID               LookAheadBuffer,
51      IN  UINT                LookaheadBufferSize,
52      IN  UINT                PacketSize);
53
54  VOID
55  CLReceiveComplete(
56      IN  NDIS_HANDLE         ProtocolBindingContext);
57
58  NDIS_STATUS
59  MPTransferData(
60      OUT PNDIS_PACKET        Packet,
61      OUT PUINT               BytesTransferred,
62      IN  NDIS_HANDLE         MiniportAdapterContext,
63      IN  NDIS_HANDLE         MiniportReceiveContext,
64      IN  UINT                ByteOffset,
65      IN  UINT                BytesToTransfer);
66
67  VOID
68  CLTransferDataComplete(
69      IN  NDIS_HANDLE         ProtocolBindingContext,
70      IN  PNDIS_PACKET        pNdisPacket,
71      IN  NDIS_STATUS         Status,
72      IN  UINT                BytesTransferred);
73
74
75  VOID
76  MPReturnPacket(
77      IN  NDIS_HANDLE         MiniportAdapterContext,
78      IN  PNDIS_PACKET        Packet)
79  {
80      PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
81      PTNS_PACKET_CONTEXT PktContext;
82      PNDIS_PACKET MPPacket;
```

```
 83      PNDIS_BUFFER NdisBuffer;
 84      PBUFFER_CONTEXT BufContext;
 85      UINT Length;
 86      PUCHAR MediaArea;
 87      UINT Size;
 88
 89      DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPReturnPackets =>\n"));
 90
 91      //
 92      //See if the Original Packet field indicates that this belongs
 93      //to someone below us and return it now
 94      //
 95
 96      PktContext = PACKET_CONTEXT_FROM_PACKET( Packet );
 97
 98      MPPacket = PktContext->OriginalPacket;
 99
100      DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "(%08X) MPReturnPacket: IM Packet %08X\n", pAdapter, Packet));
101
102      if ( MPPacket ) {
103
104          D((0, "(%08X) MPReturnPacket: Returning MP Packet %08X\n", pAdapter, Packet));
105
106          NdisReturnPackets( &MPPacket, 1 );
107
108      } else {
109
110          //
111          //It's media specific access was allocated, free it now
112          //
113          NDIS_GET_PACKET_MEDIA_SPECIFIC_INFO( Packet, &MediaArea, &Size );
114
115          NdisUnchainBufferAtFront( Packet, &NdisBuffer );
116
117          MyAssert( NdisBuffer != NULL );
118
119          NdisQueryBuffer( NdisBuffer, &BufContext, &Length );
120
121          NdisFreeBuffer(NdisBuffer);
122          NdisFreeMemory(BufContext, Length, 0);
123
124          NdisUnchainBufferAtFront( Packet, &NdisBuffer );
125
126          if ( NdisBuffer ) {
127              BreakPoint();
128          }
129      }
130
131      NdisReinitializePacket( Packet );
132      NdisFreePacket(Packet);
133
134      DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPReturnPackets <=\n"));
135 } //MPReturnPacket
136
137 unsigned char BroadcastAddress[] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
138
139 int
140 TnsCheckAddressEtherType(
141     PADAPTER pAdapter,
142     unsigned char *pHeaderBuffer,
143     ULONG    HeaderBufferSize)
144 {
145     int bcast = FALSE;
146     int ucast = FALSE;
147     unsigned short *pEtherType;
148
149     //
150     //See if this packet is a broadcast
151     //
152     if (memcmp(pHeaderBuffer, BroadcastAddress, 6) == 0) {
153         bcast = TRUE;
154         //
155         //any address is dropped back this is the same for us
156         //broadcast, then do not process the packet
157         //
158         if (memcmp(&pHeaderBuffer[6], pAdapter->LowerMPMacAddress, 6) == 0) {
159             return FALSE;
160         }
161     }
162     //
163     //Get a pointer to the ethertype
164     //
```

**File: D:\nt4DDK\src\tlmesn\tnsdrvr\recv.c**           **Page 3 of 12**

```
165      pEtherType = (unsigned short *)&pHeaderBuffer[12];
166
167      //
168      ███████████████████████████████
169      //
170      if ( TNS_EMULATION_ETHERTYPE== wswap(*pEtherType) ) {
171          return TRUE;
172      }
173
174      //
175      ███████████████████████████████
176      //
177      return FALSE;
178 }
179
180
181 NDIS_STATUS
182 CLReceiveIndication(
183      IN  NDIS_HANDLE      ProtocolBindingContext,
184      IN  NDIS_HANDLE      MacReceiveContext,
185      IN  PVOID            HeaderBuffer,
186      IN  UINT             HeaderBufferSize,
187      IN  PVOID            LookaheadBuffer,
188      IN  UINT             LookaheadBufferSize,
189      IN  UINT             PacketSize)
190 {
191      PADAPTER             pAdapter = (PADAPTER)ProtocolBindingContext;
192      PSINGLE_LIST_ENTRY   ResidualEntry = NULL;
193      PTNS_PACKET_CONTEXT  PktContext;
194      PNDIS_BUFFER         LookaheadNdisBuffer;
195      PNDIS_PACKET         OurPacket;
196      NDIS_STATUS          Status;
197      NDIS_STATUS          OurPacketStatus=NDIS_STATUS_SUCCESS;
198      PVOID                vBuffer;
199      NDIS_PHYSICAL_ADDRESS HighAddress = NDIS_PHYSICAL_ADDRESS_CONST( -1, -1 );
200      int i;
201
202      DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveIndication =>\n"));
203
204
205      if (!pAdapter->TNSDriverInitialized) {
206          //
207          ████████████████████████████████████████████████████
208          //
209          BreakPoint();
210          return NDIS_STATUS_NOT_ACCEPTED;
211      }
212
213
214      //
215      ████████████████████████████
216      //
217      if (HeaderBufferSize >= 14) {
218          if (TnsCheckAddressEtherType(pAdapter, HeaderBuffer, HeaderBufferSize)) {
219              unsigned short *pEtherType;
220              PVOID pTnsPacket = NULL;
221              PTNSPacketHeader pTnsPacketHeader = NULL;
222              unsigned short TNSCommand;
223              //
224              ███████████████████████████
225              //
226              //
227              ████████████████████████████████████████████
228              ████████████████████████
229
230              ████████████████████████████████████
231
232              if (HeaderBufferSize == PacketSize) {
233                  pTnsPacket = HeaderBuffer;
234              }
235              if ((pTnsPacket == NULL) & (HeaderBufferSize < PacketSize) ) {
236                  if (HeaderBufferSize == 14) {
237                      pTnsPacket = &((unsigned char *)LookaheadBuffer)[-14];
238                  }
239              }
240              ████████████████████████████████████████████████████████████████████████████████████
 -2  ██████████████████████████████████████████
241
242              MyAssert(pTnsPacket != NULL);
243
244              ████████████████████████████████████████████████████
245
```

```
246              TNSCommand = wswap(((PTNSPacketHeader)pTnsPacket)->TNSCommandReply);
247
248              switch (TNSCommand) {
249                  case TNS_HELLO_BROADCAST:
250                      D((0, "TNS_HELLO_BROADCAST\n"));
251                      if (TNSSharedMemoryNodeEmulation) {
252                          // prepare the num emul stpr,
253                          // build a reply
254                          TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvHelloBroadcasts);
255
256                          //
257                          // Only process the reply if the shared memory region has
258                          // been allocated.
259                          //
260                          if ( (pAdapter->TNSSharedMemoryPtr) && (pAdapter->TNSSharedMemorySize) ) {
261                              TNSBuildBroadcastReplyAndSend(pAdapter, pTnsPacket, HeaderBuffer);
262                          }
263                      } else {
264                          //
265                          // else just drop the broadcast packets
266                          // are looped back;
267                      }
268                      break;
269                  case TNS_HELLO_REPLY:
270                      D((0, "TNS_HELLO_REPLY\n"));
271                      if (TNSSharedMemoryNodeEmulation) {
272                          //
273                          // we are emulating the same host path
274                          // this request
275                          //
276                          MyAssert(0);
277                      } else {
278                          PLIST_ENTRY pRequestObj;
279                          PREQUEST_DATA pRqstData;
280                          unsigned char *pBuffer;
281                          //
282                          // we need to set our node ID and the
283                          // smn mac address;
284                          //
285                          pAdapter->TNSClientNodeID = ((PTNSPacketHelloReply)pTnsPacket)->TNSClientNodeID;
286                          D((0, "Server Hello reply, Client NodeID => %d\n", pAdapter->TNSClientNodeID));
287                          pAdapter->TNSSharedMemorySize = dwswap(((PTNSPacketHelloReply)pTnsPacket)->TNSSha
 -2 redMemorySize);
288
289                          D((0, "TNSSharedMemorySize => %x\n", pAdapter->TNSSharedMemorySize));
290
291                          for (i=0; i<6; i++) {
292                              pAdapter->SMNMacAddress[i] = ((PTNSPacketHelloReply)pTnsPacket)->SMNServerMac
 -2 Address[i];
293                          }
294                          RtlCopyMemory(&pAdapter->SMNMachineName, ((PTNSPacketHelloReply)pTnsPacket)->SMNM
 -2 achineName, 16);
295                          //
296                          // Dequeue a free element from our available object queue
297                          //
298                          pRequestObj = ExInterlockedRemoveHeadList(
299                              &pAdapter->WorkerListEntryPool,
300                              &pAdapter->ListEntryPoolLock);
301
302                          pRqstData = CONTAINING_RECORD(pRequestObj,
303                                      REQUEST_DATA,
304                                      Linkage);
305
306                          //
307                          // set up the request data object
308                          //
309                          pRqstData->pNdisPacket = NULL;
310                          pRqstData->requestOpcode = TNS_HELLO_REPLY;
311
312                          //
313                          // insert into the client worker list queue
314                          //
315                          ExInterlockedInsertTailList(
316                              &pAdapter->ClientWorkerListEntry,
317                              &pRqstData->Linkage,
318                              &pAdapter->ClientWorkerListSpinLock);
319
320                          //
321                          // release the server worker
322                          //
323                          KeReleaseSemaphore(
324                              &pAdapter->ClientWorkerResponseSemaphore,
```

117

```
325                               (KPRIORITY) 0,
326                               (LONG) 1,
327                               FALSE);
328                          //
329                          //We need to process this as complete
330                          //
331
332                      }
333                      break;
334                  case TNS_READ_REQUEST:
335                      //case TNS_READ_REQUEST\\\\\
336                      if (TNSSharedMemoryNodeEmulation) {
337                          PLIST_ENTRY pRequestObj;
338                          PREQUEST_DATA pRqstData;
339                          unsigned char *pBuffer;
340
341                          TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvReadRequests);
342
343                          if (pAdapter->TNSMemoryType == VIRTUAL_MEMORY) {
344
345                              //
346                              //We need to service this read request
347                              //
348
349                              //
350                              //Dequeue a free element from our available object queue
351                              //
352                              pRequestObj = ExInterlockedRemoveHeadList(
353                                  &pAdapter->WorkerListEntryPool,
354                                  &pAdapter->ListEntryPoolLock);
355
356                              MyAssert(pRequestObj);
357
358                              pRqstData = CONTAINING_RECORD(pRequestObj,
359                                          REQUEST_DATA,
360                                          Linkage);
361
362                              MyAssert(pRqstData);
363
364                              //
365                              //Copy the necessary thread data into the
366                              //
367                              pRqstData->pNdisPacket = NULL;
368                              pRqstData->requestOpcode = TNS_READ_REQUEST;
369                              pBuffer = (unsigned char *)&pRqstData->TnsPacket;
370                              RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
371                              RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSiz
 -2 e);
372
373                              //
374                              //Insert object onto server thread object queue
375                              //
376                              ExInterlockedInsertTailList(
377                                  &pAdapter->ServerWorkerListEntry,
378                                  &pRqstData->Linkage,
379                                  &pAdapter->ServerWorkerListSpinLock);
380
381                              //
382                              //Now signal the server thread
383                              //
384                              KeReleaseSemaphore(
385                                  &pAdapter->ServerWorkerRequestSemaphore,
386                                  (KPRIORITY) 0,
387                                  (LONG) 1,
388                                  FALSE);
389                          }
390
391                          if (pAdapter->TNSMemoryType == NONPAGED_MEMORY) {
392                              PNDIS_PACKET MyPacket;
393                              ULONG PacketLength;
394                              PVOID pTnsBuffer;
395                              NTSTATUS Status;
396                              PUCHAR  vBuffer;
397
398                              vBuffer = pAdapter->TNSSharedMemoryPtr;
399
400                              PacketLength = TNS_PACKET_SIZE(TNSPacketReadReply);
401
402                              Status = TNSInitializeClientNodeSendPacket(pAdapter,
403                                  &MyPacket,
404                                  &pTnsBuffer,
405                                  PacketLength);
```

```
406
407                                RtlCopyMemory(pTnsBuffer, &((PTNSPacketHeader)pTnsPacket)->MACSrcAddress, 6);
408                                //
409                                // Store relavent packet information here
410                                //
411                                ((PTNSPacketHeader)pTnsBuffer)->TNSCommandReply = wswap(TNS_READ_REPLY);
412
413                                ((PTNSPacketReadReply)pTnsBuffer)->RequestTag       = ((PTNSPacketReadRequest
-2  )pTnsPacket)->RequestTag;
414                                ((PTNSPacketReadReply)pTnsBuffer)->RequestStartTSC  = ((PTNSPacketReadRequest
-2  )pTnsPacket)->RequestStartTSC;
415                                vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap(((PTNSPacketReadRequest)pTnsP
-2  acket)->RequestOffset));
416
417                                if (dwswap(((PTNSPacketReadRequest)pTnsPacket)->RequestOffset) <= pAdapter->T
-2  NSSharedMemorySize ) {
418                                    ((PTNSPacketReadReply)pTnsBuffer)->dwData = *((PULONG)vBuffer);
419                                } else {
420                                    _asm int 3
421                                }
422                                TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
423                            }
424
425                        } else {
426                            MyAssert(0);
427                        }
428                        break;
429                    case TNS_READ_REPLY:
430                        // case TNS_READ_REPLY:
431                        if (TNSSharedMemoryNodeEmulation) {
432                            //
433                            // node emulation first copy this
434                            //
435                            MyAssert(0);
436                        } else {
437                            PLIST_ENTRY pRequestObj;
438                            PREQUEST_DATA pRqstData;
439                            unsigned char *pBuffer;
440                            //
441                            // need to service this read request
442
443
444                            //
445                            // dequeue a element from our available object queue
446                            //
447                            pRequestObj = ExInterlockedRemoveHeadList(
448                                &pAdapter->WorkerListEntryPool,
449                                &pAdapter->ListEntryPoolLock);
450
451                            pRqstData = CONTAINING_RECORD(pRequestObj,
452                                        REQUEST_DATA,
453                                        Linkage);
454
455                            //
456                            // setup the worker thread what to do
457                            //
458                            pRqstData->pNdisPacket = NULL;
459                            pRqstData->requestOpcode = TNS_READ_REPLY;
460                            pBuffer = (unsigned char *)&pRqstData->TnsPacket;
461                            RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
462                            RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);
463
464.                           //
465                            // insert object into the worker thread request queue
466                            //
467                            ExInterlockedInsertTailList(
468                                &pAdapter->ClientWorkerListEntry,
469                                &pRqstData->Linkage,
470                                &pAdapter->ClientWorkerListSpinLock);
471
472                            //
473                            // signal the worker thread
474                            //
475                            KeReleaseSemaphore(
476                                &pAdapter->ClientWorkerRequestSemaphore,
477                                (KPRIORITY) 0,
478                                (LONG) 1,
479                                FALSE);
480                            //
481                            // we are processing the packet
482                            //
483                        }
```

```
484                   break;
485             case TNS_WRITE_REQUEST:
486                   //DEBUG("TNS_WRITE_REQUEST\n");
487
488                   if (TNSSharedMemoryNodeEmulation) {
489
490                         TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvWriteRequests);
491
492                         if (pAdapter->TNSMemoryType == VIRTUAL_MEMORY) {
493                               //
494                               // Process write request on the host
495                               //
496                               PLIST_ENTRY pRequestObj;
497                               PREQUEST_DATA pRqstData;
498                               unsigned char *pBuffer;
499                               //
500                               // We need to service this read request
501                               //
502
503                               //
504                               // Dequeue carrier element from our available object queue
505                               //
506                               pRequestObj = ExInterlockedRemoveHeadList(
507                                     &pAdapter->WorkerListEntryPool,
508                                     &pAdapter->ListEntryPoolLock);
509
510                               pRqstData = CONTAINING_RECORD(pRequestObj,
511                                           REQUEST_DATA,
512                                           Linkage);
513
514                               //
515                               // Fill in request information record
516                               //
517                               pRqstData->pNdisPacket = NULL;
518                               pRqstData->requestOpcode = TNS_WRITE_REQUEST;
519                               pBuffer = (unsigned char *)&pRqstData->TnsPacket;
520                               RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
521                               RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSiz
 -2 e);
522
523                               //
524                               // Insert this request onto our server worker object queue
525                               //
526                               ExInterlockedInsertTailList(
527                                     &pAdapter->ServerWorkerListEntry,
528                                     &pRqstData->Linkage,
529                                     &pAdapter->ServerWorkerListSpinLock);
530
531                               //
532                               // Wake up a server worker thread
533                               //
534                               KeReleaseSemaphore(
535                                     &pAdapter->ServerWorkerRequestSemaphore,
536                                     (KPRIORITY) 0,
537                                     (LONG) 1,
538                                     FALSE);
539                         }
540
541                         if (pAdapter->TNSMemoryType == NONPAGED_MEMORY) {
542
543                               PNDIS_PACKET MyPacket;
544                               ULONG PacketLength;
545                               PVOID pTnsBuffer;
546                               NTSTATUS Status;
547                               PUCHAR   vBuffer;
548
549                               // Process write request on the server
550
551                               vBuffer = pAdapter->TNSSharedMemoryPtr;
552
553                               vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap( ((PTNSPacketWriteRequest)pTn
 -2 sPacket)->RequestOffset));
554
555                               if (dwswap( ((PTNSPacketWriteRequest)pTnsPacket)->RequestOffset) <= pAdapter-
 -2 >TNSSharedMemorySize ) {
556                                     *((PULONG)vBuffer) = ((PTNSPacketWriteRequest)pTnsPacket)->dwData;
557                               } else {
558                                     _asm int 3
559                               }
560
561                               //
562                               // Service complete
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c**　　　　　　　**Page 8 of 12**

```
563                              ////
564
565                              PacketLength = TNS_PACKET_SIZE(TNSPacketWriteReply);
566                              Status = TNSInitializeClientNodeSendPacket(pAdapter,
567                                  &MyPacket,
568                                  &pTnsBuffer,
569                                  PacketLength);
570
571                              RtlCopyMemory(pTnsBuffer, &((PTNSPacketWriteRequest)pTnsPacket)->MACSrcAddres
  -2 s, 6);
572                              ////
573                              ////
574                              ////
575                              ((PTNSPacketWriteReply)pTnsBuffer)->TNSCommandReply = wswap(TNS_WRITE_ACK);
576                              ((PTNSPacketWriteReply)pTnsBuffer)->RequestTag      = ((PTNSPacketWriteReques
  -2 t)pTnsPacket)->RequestTag;
577                              ((PTNSPacketWriteReply)pTnsBuffer)->RequestStartTSC = ((PTNSPacketWriteReques
  -2 t)pTnsPacket)->RequestStartTSC;
578
579                              TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
580                          }
581
582                  } else {
583                          ////
584                          ////
585                          ////
586                          MyAssert(0);
587                  }
588                  break;
589
590          case TNS_WRITE_ACK:
591                  ////
592                  if (TNSSharedMemoryNodeEmulation) {
593                          ////
594                          ////
595                          ////
596                          MyAssert(0);
597                  } else {
598                          PLIST_ENTRY pRequestObj;
599                          PREQUEST_DATA pRqstData;
600                          unsigned char *pBuffer;
601                          ////
602                          ////
603                          ////
604
605                          ////
606                          ////
607                          ////
608                          pRequestObj = ExInterlockedRemoveHeadList(
609                              &pAdapter->WorkerListEntryPool,
610                              &pAdapter->ListEntryPoolLock);
611
612                          pRqstData = CONTAINING_RECORD(pRequestObj,
613                                  REQUEST_DATA,
614                                  Linkage);
615
616                          ////
617                          ////
618                          ////
619                          pRqstData->pNdisPacket = NULL;
620                          pRqstData->requestOpcode = TNS_WRITE_ACK;
621                          pBuffer = (unsigned char *)&pRqstData->TnsPacket;
622                          RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
623                          RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);
624
625                          ////
626                          ////
627                          ////
628                          ExInterlockedInsertTailList(
629                              &pAdapter->ClientWorkerListEntry,
630                              &pRqstData->Linkage,
631                              &pAdapter->ClientWorkerListSpinLock);
632
633                          ////
634                          ////
635                          ////
636                          KeReleaseSemaphore(
637                              &pAdapter->ClientWorkerRequestSemaphore,
638                              (KPRIORITY) 0,
639                              (LONG) 1,
640                              FALSE);
641                          ////
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c**

```
642                                 //We need to process this as complete
643                                 //
644                         }
645                     break;
646                 case TNS_QUERY_STATS: {
647                         //DX1016 TNS_READ_REQUEST_DATA
648                         PLIST_ENTRY pRequestObj;
649                         PREQUEST_DATA pRqstData;
650                         unsigned char *pBuffer;
651
652                         PNDIS_PACKET MyPacket;
653                         ULONG PacketLength;
654                         PTNSPacketQueryStatsReply pTnsBuffer;
655                         NTSTATUS Status;
656                         NDIS_STATUS NdisStatus;
657                         PUCHAR   vBuffer;
658
659                         TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvQueryStats);
660
661                         vBuffer = pAdapter->TNSSharedMemoryPtr;
662
663                         PacketLength = TNS_PACKET_SIZE(TNSPacketQueryStatsReply);
664
665                         Status = TNSInitializeClientNodeSendPacket(pAdapter,
666                             &MyPacket,
667                             &pTnsBuffer,
668                             PacketLength);
669
670                         RtlCopyMemory(pTnsBuffer, &((PTNSPacketHeader)pTnsPacket)->MACSrcAddress, 6);
671                         //
672                         //Set up the event packet information here
673                         //
674                         pTnsBuffer->TNSCommandReply = wswap(TNS_QUERY_STATS_REPLY);
675
676                         pTnsBuffer->RequestTag = ((PTNSPacketQueryStats)pTnsPacket)->RequestTag;
677                         pTnsBuffer->RequestStartTSC = ((PTNSPacketQueryStats)pTnsPacket)->RequestStartTSC
 -2 ;
678
679                         RtlCopyMemory(&pTnsBuffer->TnsNodeStatistics, &pAdapter->MyStats, sizeof(STATISTI
 -2 CS) );
680                         RtlCopyMemory(&pTnsBuffer->MpStats, &pAdapter->mpStats, sizeof(MPSTATS) );
681
682                         pTnsBuffer->NdisStatus = STATUS_SUCCESS;
683
684                         TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
685
686                     }
687                 break;
688
689             case TNS_CLEAR_STATS:
690                     //DX1016 TNS_READ_REQUEST_DATA
691
692                     RtlZeroMemory(&pAdapter->MyStats, sizeof(STATISTICS) );
693                     RtlZeroMemory(&pAdapter->mpStats, sizeof(MPSTATS) );
694
695                     break;
696
697             case TNS_QUERY_STATS_REPLY: {
698                     //DX1016 TNS_READ_REQUEST_DATA
699                     PLIST_ENTRY pRequestObj;
700                     PREQUEST_DATA pRqstData;
701                     unsigned char *pBuffer;
702                     //
703                     //
704                     //
705
706                     //
707                     //Remove the data element from the Worker queue
708                     //
709                     pRequestObj = ExInterlockedRemoveHeadList(
710                         &pAdapter->WorkerListEntryPool,
711                         &pAdapter->ListEntryPoolLock);
712
713                     pRqstData = CONTAINING_RECORD(pRequestObj,
714                                 REQUEST_DATA,
715                                 Linkage);
716
717                     //
718                     //
719                     //
720                     pRqstData->pNdisPacket = NULL;
721                     pRqstData->requestOpcode = TNS_QUERY_STATS_REPLY;
```

```
722                        pBuffer = (unsigned char *)&pRqstData->TnsPacket;
723                        RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
724                        RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);
725
726                        //
727                        // Insert object onto server thread object queue
728                        //
729                        ExInterlockedInsertTailList(
730                            &pAdapter->ClientWorkerListEntry,
731                            &pRqstData->Linkage,
732                            &pAdapter->ClientWorkerListSpinLock);
733
734                        //
735                        // Now signal the server thread
736                        //
737                        KeReleaseSemaphore(
738                            &pAdapter->ClientWorkerRequestSemaphore,
739                            (KPRIORITY) 0,
740                            (LONG) 1,
741                            FALSE);
742                        //
743                        // No need to process this as complete
744                        //
745                    }
746                    break;
747
748                case TNS_STRING_WRITE_REQUEST:
749                    D((0, "TNS_STRING_WRITE_REQUEST\n"));
750                    MyAssert(0);
751                    if (TNSSharedMemoryNodeEmulation) {
752                    } else {
753                    }
754                    break;
755                case TNS_STRING_READ_REQUEST:
756                    D((0, "TNS_STRING_READ_REQUEST\n"));
757                    MyAssert(0);
758                    if (TNSSharedMemoryNodeEmulation) {
759                    } else {
760                    }
761                    break;
762                case TNS_STRING_READ_REPLY:
763                    D((0, "TNS_STRING_READ_REPLY\n"));
764                    MyAssert(0);
765                    if (TNSSharedMemoryNodeEmulation) {
766                    } else {
767                    }
768                    break;
769                default:
770                    D((0, "Unrecognized command => %x\n", TNSCommand));
771                    D((0, "HeaderBuffer    => %x, HdrBufferSize => %x\n", HeaderBuffer, HeaderBufferSize))
-2  ;
772                    D((0, "LookahedBuffer => %x, LABufferSize  => %x\n", LookaheadBuffer, LookaheadBuffer
-2  Size));
773                    MyAssert(0);
774                    break;
775                }
776                // return NDIS_STATUS_SUCCESS;
777            }
778
779        } else {
780            D((0, "HeaderBufferSize not equal to or gt than 14, HeaderBufferSize => %d\n", HeaderBufferSize))
-2  ;
781            _asm int 3
782        }
783
784    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "HeaderBuffer => %x, HeaderBufferSize => %x, LookaheadBuffer =>
-2  %x, LookaheadBufferSize => %x\n",
785            HeaderBuffer,
786            HeaderBufferSize,
787            LookaheadBuffer,
788            LookaheadBufferSize));
789
790    NdisAllocatePacket(&Status, &OurPacket, pAdapter->PacketPoolHandle);
791
792    NdisReinitializePacket(OurPacket);
793
794    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "CLReceiveIndication: OurPacket => %x\n", OurPacket));
795
796    MyAssert(OurPacket->Private.Head == NULL);
797
798    NDIS_SET_PACKET_STATUS(OurPacket, OurPacketStatus);
799
```

```
800      Status = NdisAllocateMemory(&vBuffer, 2000, 0, HighAddress);
801
802      if (Status != NDIS_STATUS_SUCCESS) {
803          BreakPoint();
804      }
805
806      NdisAllocateBuffer(&Status,
807          &LookaheadNdisBuffer,
808          pAdapter->LookaheadPoolHandle,
809          vBuffer,
810          2000);
811
812      if (Status != NDIS_STATUS_SUCCESS) {
813          BreakPoint();
814      }
815
816      DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "CLReceiveIndication: LookaheadNdisBuffer => %x\n", LookaheadNd
-2 isBuffer));
817
818      PktContext = PACKET_CONTEXT_FROM_PACKET(OurPacket);
819
820      DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "(%08X) CLReceiveIndication: Packet %08X Packetsize %d %s\n",
821          pAdapter, OurPacket, PacketSize,
822          (PacketSize != LookaheadBufferSize ? "(RD)" : "")));
823
824      PktContext->OriginalPacket = NULL;
825
826      if (pAdapter->CopyLookaheadData) {
827          NdisMoveMemory(vBuffer, HeaderBuffer, HeaderBufferSize);
828          NdisMoveMemory((CHAR *)vBuffer+HeaderBufferSize, LookaheadBuffer, LookaheadBufferSize);
829      } else {
830          TdiCopyLookaheadData(vBuffer, HeaderBuffer, HeaderBufferSize, 0);
831          TdiCopyLookaheadData((CHAR *)vBuffer+HeaderBufferSize, LookaheadBuffer, LookaheadBufferSize, 0);
832      }
833
834      NdisAdjustBufferLength(LookaheadNdisBuffer, HeaderBufferSize+LookaheadBufferSize);
835      NDIS_SET_PACKET_HEADER_SIZE(OurPacket, HeaderBufferSize);
836      NdisChainBufferAtFront(OurPacket, LookaheadNdisBuffer);
837
838      DUMP_PACKET(OurPacket);
839
840      DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "Adapter->TNSNdisHandle => %x, OurPacket => %x\n", pAdapter->TN
-2 SNdisHandle, OurPacket));
841      NDIS_SET_PACKET_STATUS(OurPacket, NDIS_STATUS_RESOURCES);
842
843      NdisMIndicateReceivePacket(pAdapter->TNSNdisHandle, &OurPacket, 1);
844
845      if ( NDIS_GET_PACKET_STATUS(OurPacket) != NDIS_STATUS_PENDING) {
846          MPReturnPacket( (NDIS_HANDLE)pAdapter, OurPacket );
847      }
848
849      DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveIndication <=\n"));
850      return NDIS_STATUS_SUCCESS;
851
852 } // CLReceiveIndication
853
854
855 VOID
856 CLReceiveComplete(
857      IN  NDIS_HANDLE      ProtocolBindingContext)
858 {
859      PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
860
861 .    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveComplete =>\n"));
862
863      if (pAdapter->TNSDriverInitialized) {
864
865          switch( pAdapter->MediaType ) {
866              case NdisMedium802_3:
867                  DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "(%08X) CLReceiveComplete: 802_3\n", pAdapter));
868                  NdisMEthIndicateReceiveComplete( pAdapter->TNSNdisHandle );
869                  break;
870
871              case NdisMedium802_5:
872                  D((0, "(%08X) CLReceiveComplete: 802_5\n", pAdapter));
873                  BreakPoint();
874                  NdisMTrIndicateReceiveComplete( pAdapter->TNSNdisHandle );
875                  break;
876
877              case NdisMediumFddi:
878                  D((0, "(%08X) CLReceiveComplete: FDDI\n", pAdapter));
879                  BreakPoint();
```

```
880                NdisMFddiIndicateReceiveComplete( pAdapter->TNSNdisHandle );
881                break;
882
883            default:
884                MyAssert( FALSE );
885        }
886    } else {
887        BreakPoint();
888    }
889
890    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveComplete <=\n"));
891 }  //CLReceiveComplete
892
893 NDIS_STATUS
894 MPTransferData(
895    OUT PNDIS_PACKET        Packet,
896    OUT PUINT               BytesTransferred,
897    IN  NDIS_HANDLE         MiniportAdapterContext,
898    IN  NDIS_HANDLE         MiniportReceiveContext,
899    IN  UINT                ByteOffset,
900    IN  UINT                BytesToTransfer)
901 {
902    PADAPTER Adapter = (PADAPTER)MiniportAdapterContext;
903
904    D((0, "(%08X) MPTransferData:\n", Adapter));
905    BreakPoint();
906    return NDIS_STATUS_FAILURE;
907 }  //MPTransferData
908
909 VOID
910 CLTransferDataComplete(
911    IN  NDIS_HANDLE         ProtocolBindingContext,
912    IN  PNDIS_PACKET        Packet,
913    IN  NDIS_STATUS         Status,
914    IN  UINT                BytesTransferred)
915 {
916    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
917    PTNS_PACKET_CONTEXT PktContext;
918
919    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLTransferComplete =>\n"));
920    D((0, "(%08X) CLTransferDataComplete: Packet %08X Status %08X Bytes xfer'ed %d\n",
921        pAdapter, Packet, Status, BytesTransferred));
922
923    PktContext = PACKET_CONTEXT_FROM_PACKET( Packet );
924
925    NdisChainBufferAtFront(Packet, PktContext->LookaheadBuffer);
926
927    NdisMIndicateReceivePacket(pAdapter->TNSNdisHandle, &Packet, 1);
928
929    if ( NDIS_GET_PACKET_STATUS(Packet) != NDIS_STATUS_PENDING) {
930        MPReturnPacket((NDIS_HANDLE)pAdapter, Packet);
931    }
932
933    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLTransferComplete <=\n"));
934 }  //CLTransferDataComplete
935
936
```

**File: D:\nt4DDK\src\tlmesn\tnsdrvr\send.c**                   **Page 1 of 3**

```
33  #include "tns.h"
34  #include "tnsdebug.h"
35  #include "x86.h"
36
37  #define MAX_LOCAL_PACKET_ARRAY  10
38
39  VOID
40  MPSendPackets(
41      IN  NDIS_HANDLE MiniportAdapterContext,
42      IN  PPNDIS_PACKET PacketArray,
43      IN  UINT NumberOfPackets
44      );
45
46
47  VOID
48  CLSendComplete(
49      IN  NDIS_HANDLE ProtocolBindingContext,
50      IN  PNDIS_PACKET Packet,
51      IN  NDIS_STATUS Status
52      );
53
54  VOID
55  MPSendPackets(
56·     IN  NDIS_HANDLE     MiniportAdapterContext,
57      IN  PPNDIS_PACKET   PacketArray,
58      IN  UINT            NumberOfPackets)
59  {
60      PADAPTER            pAdapter=(PADAPTER)MiniportAdapterContext;
61      PNDIS_PACKET        Packet;
62      PNDIS_PACKET        MyPacket;
63      PNDIS_PACKET        MyPacketArray[MAX_LOCAL_PACKET_ARRAY];
64
65      PSINGLE_LIST_ENTRY      PacketEntry = NULL;
66      PTNS_PACKET_CONTEXT     PktContext;
67      PNDIS_BUFFER    FirstBuffer;
68      PNDIS_PACKET_OOB_DATA   MyOOBData;
69      PNDIS_PACKET_OOB_DATA   OOBData;
70      ULONG       PacketLength, i;
71      ULONG       NumMyPackets=0;
72      NDIS_STATUS     Status;
73
74      DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPSendPackets ->\n"));
75      DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "(%08X) MPSendPackets: %d XPORT packets\n", pAdapter, Numb
  -2 ackets));
76
77      if (pAdapter) {
78          if (!pAdapter->TNSDriverInitialized) {
79
80
81
```

```
 82                   BreakPoint();
 83            }
 84      }
 85
 86      for (i=0; i<NumberOfPackets; ++i) {        //
 87
 88          //
 89
 90          Packet = PacketArray[i];
 91
 92          DUMP_PACKET(Packet);
 93          /*
 94          //
 95          //
 96          */
 97          NdisAllocatePacket(&Status, &MyPacket, pAdapter->PacketPoolHandle);
 98
 99          /*
100          //
101          */
102          MyAssert(MyPacket->Private.Head == NULL);
103
104          PktContext = PACKET_CONTEXT_FROM_PACKET(MyPacket);
105
106          DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "MPSendPackets: MyPacket => %x\n", PacketEntry));
107
108
109          NdisQueryPacket(Packet, NULL, NULL, &FirstBuffer, &PacketLength);
110
111          NdisChainBufferAtFront(MyPacket, FirstBuffer);
112
113          NdisSetPacketFlags(MyPacket, NdisGetPacketFlags(Packet));
114
115          OOBData = NDIS_OOB_DATA_FROM_PACKET(Packet);
116          MyOOBData = NDIS_OOB_DATA_FROM_PACKET(MyPacket);
117          NdisMoveMemory(MyOOBData, OOBData, sizeof(NDIS_PACKET_OOB_DATA));
118
119          /*
120          //
121          */
122          NDIS_SET_PACKET_STATUS(Packet, NDIS_STATUS_PENDING);
123
124          /*
125          //
126          */
127          /*
128          PktContext->OriginalPacket = Packet;
129          PktContext->SMNEmulationPacket = FALSE;
130
131          /*
132          //
133          */
134          DUMP_PACKET(MyPacket);
135          MyPacketArray[NumMyPackets++] = MyPacket;
136      }
137
138      if (NumMyPackets) {
139          int FoundFlag;
140          for (i=0; i<NumMyPackets; i++) {
141              DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "MPSendPackets, Packet Status => %x, %s\n",
142              NDIS_GET_PACKET_STATUS(MyPacketArray[i]),
143              GetNDISStatusString(NDIS_GET_PACKET_STATUS(MyPacketArray[i]), &FoundFlag) ));
144          }
145          NdisSendPackets(pAdapter->LowerMPHandle, &MyPacketArray[0], NumMyPackets);
146      }
147
148      DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPSendPackets <=\n"));
149 }
150
151 int printbuftime = 1;
152
153 VOID
154 CLSendComplete(
155      IN  NDIS_HANDLE      ProtocolBindingContext,
156      IN  PNDIS_PACKET     Packet,
157      IN  NDIS_STATUS      Status)
158 {
159      PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
160      PTNS_PACKET_CONTEXT PktContext;
161      int FoundFlag;
162      int SMNEmulationPacket;
163      PNDIS_BUFFER MyBuffer;
```

```
164    PTNSPacketReadRequest BufContext;
165    UINT Length;
166
167    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLSendComplete ->\n"));
168
169    DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "CLSendComplete, Packet Status => %x, %s\n",
170        NDIS_GET_PACKET_STATUS(Packet),
171        GetNDISStatusString(NDIS_GET_PACKET_STATUS(Packet), &FoundFlag) ));
172
173    PktContext = PACKET_CONTEXT_FROM_PACKET(Packet);
174    SMNEmulationPacket = PktContext->SMNEmulationPacket;
175
176    DUMP_PACKET(Packet);
177    if (PktContext->OriginalPacket) {
178        DUMP_PACKET(PktContext->OriginalPacket);
179        DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "CLSendComplete, Packet Status => %x, %s\n",
180            NDIS_GET_PACKET_STATUS(PktContext->OriginalPacket),
181            GetNDISStatusString(NDIS_GET_PACKET_STATUS(PktContext->OriginalPacket), &FoundFlag) ));
182    }
183
184    if (SMNEmulationPacket) {
185        NdisUnchainBufferAtFront(Packet, &MyBuffer);
186        NdisQueryBuffer(MyBuffer, &BufContext, &Length);
187        NdisFreeBuffer(MyBuffer);
188        NdisFreeMemory(BufContext, Length, 0);
189    }
190
191    // Because most have to unchain and free the buffer allocated in CLSend
192    NdisReinitializePacket(Packet);
193    NdisFreePacket(Packet);
194    // CLSendComplete the DISG packet
195    //
196
197    if (SMNEmulationPacket == FALSE) {
198        NdisMSendComplete(pAdapter->TNSNdisHandle, PktContext->OriginalPacket, Status);
199    }
200
201    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLSendComplete <=\n"));
202 }  // CLSendComplete
203
204
205
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.h**

```
 1  //
 2  //
 3  // COPYRIGHT
 4  //      This program is an unpublished work fully protected by the United
 5  //      States copyright laws and is considered a trade secret belonging to
 6  //      Times N Systems, Inc.  To the extent that this work may be
 7  //      considered published, the following notice applies  1999 Times N
 8  //      Systems, Inc.  Any unauthorized use, reproduction, distribution,
 9  //      display, modification, or disclosure of this program is strictly
10  //      prohibited.
11  //
12  //
13  //
14  //
15  // Module:
16  //
17  // Description:
18  //
19  // Environment:
20  //      Windows NT Kernel Mode only.
21  //
22  // Exports:
23  //      See Module functions generated by script processing.
24  //
25  // Author:
26  //      Vince Bridgers
27  //      vinceb@timesn.com
28  //
29  //
30  //
31
32
33
34  //
35  // Define the various device type values.  Note that values used by Microsoft
36  // Corporation are in the range 0-32767, and 32768-65535 are reserved for use
37  // by customers.
38  //
39
40  #define FILE_DEVICE_TNSCLIENT   0x00008300
41
42
43
44  //
45  // Macro definition for defining IOCTL and FSCTL function control codes.  Note
46  // that function codes 0-2047 are reserved for Microsoft Corporation, and
47  // 2048-4095 are reserved for customers.
48  //
49
50  #define TNSCLIENT_IOCTL_INDEX   0x830
51
52
53
54
55  #define IOCTL_TNSCLIENT_HELLO           CTL_CODE(FILE_DEVICE_TNSCLIENT, \
56                                                   TNSCLIENT_IOCTL_INDEX,  \
57                                                   METHOD_BUFFERED,        \
58                                                   FILE_ANY_ACCESS)
59
60  #define IOCTL_TNSCLIENT_GET_LOCAL_STATS CTL_CODE(FILE_DEVICE_TNSCLIENT, \
61                                                   TNSCLIENT_IOCTL_INDEX+1, \
62                                                   METHOD_BUFFERED,        \
63                                                   FILE_ANY_ACCESS)
64
65  #define IOCTL_TNSCLIENT_GET_SMN_STATS   CTL_CODE(FILE_DEVICE_TNSCLIENT, \
66                                                   TNSCLIENT_IOCTL_INDEX+2, \
67                                                   METHOD_BUFFERED,        \
68                                                   FILE_ANY_ACCESS)
69
70
71  #define IOCTL_TNSCLIENT_GET_SMN_INFO    CTL_CODE(FILE_DEVICE_TNSCLIENT, \
72                                                   TNSCLIENT_IOCTL_INDEX+3, \
73                                                   METHOD_BUFFERED,        \
74                                                   FILE_ANY_ACCESS)
75
76  #define IOCTL_TNSCLIENT_GET_LOCAL_INFO  CTL_CODE(FILE_DEVICE_TNSCLIENT, \
77                                                   TNSCLIENT_IOCTL_INDEX+4, \
78                                                   METHOD_BUFFERED,        \
79                                                   FILE_ANY_ACCESS)
80
81
82  #define IOCTL_TNSCLIENT_DOTEST          CTL_CODE(FILE_DEVICE_TNSCLIENT, \
```

```
 83                                                 TNSCLIENT_IOCTL_INDEX+5,  \
 84                                                 METHOD_BUFFERED,          \
 85                                   .             FILE_ANY_ACCESS)
 86
 87 #define IOCTL_TNSCLIENT_CLEAR_STATS·  CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
 88                                                 TNSCLIENT_IOCTL_INDEX+6,  \
 89                                                 METHOD_BUFFERED,          \
 90                                                 FILE_ANY_ACCESS)
 91
 92
 93 #define IOCTL_TNSCLIENT_GET_SMN_TABLE_INFO CTL_CODE(FILE_DEVICE_TNSCLIENT, \
 94                                                 TNSCLIENT_IOCTL_INDEX+7,  \
 95                                                 METHOD_BUFFERED,          \
 96                                                 FILE_ANY_ACCESS)
 97
 98 #define IOCTL_TNSCLIENT_GET_NODE_INFO   CTL_CODE(FILE_DEVICE_TNSCLIENT, \
 99                                                 TNSCLIENT_IOCTL_INDEX+8, \
100                                                 METHOD_BUFFERED, \
101                                                 FILE_ANY_ACCESS)
102
103
104
105
106
107 #define ETHERNET_ADDRESS_LEN 6
108 #define MAX_COMPUTER_NAME_LEN    16
109
110
111
112 typedef struct _IODRIVER_PACKET {
113     int     MaxNumWrites;
114     int     MaxNumReads;
115     int     MaxNumReadWrites;
116
117     STATISTICS  Stats;
118     MPSTATS     MpStats;
119
120     unsigned char MacAddress[ETHERNET_ADDRESS_LEN];
121     unsigned char ComputerName[MAX_COMPUTER_NAME_LEN];
122     unsigned long TeamNodeID;
123     unsigned long TNSSharedMemorySize;
124
125     unsigned long TestStatus;
126
127     unsigned long DebugPrintFlag;
128     unsigned long DebugPrintMask;
129
130     SMNTableInfo  SMNInfo[MAX_TEAM_NODES];
131
132 } IO_DRIVER_PACKET, *pIO_DRIVER_PACKET;
133
134
135
136
137
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c**                    **Page 1 of 9**

```
 1 //*************************************************************
 2 //
 3 // COPYRIGHT:
 4 //      This program is an unpublished work fully protected by the United
 5 //      States copyright laws and is considered a trade secret belonging to
 6 //      Times N Systems, Inc.  To the extent that this work may be
 7 //      considered "published", the following notice applies:  1999 Times N
 8 //      Systems, Inc.  Any unauthorized use, reproduction, distribution,
 9 //      display, modification, or disclosure of this program is strictly
10 //      prohibited.
11 //
12 //*************************************************************
13 //
14 //*************************************************************
15 // Module:
16 //
17 // Description:
18 //
19 // Environment:
20 //      Windows NT Kernel Mode only.
21 //
22 // Exports:
23 //
24 // Author:
25 //      Vince Bridgers
26 //      vince@timesn.com
27 //
28 //
29 //*************************************************************
30
31 #include <ntddk.h>
32 #include <stdarg.h>
33 #include <stdio.h>
34 #include "tnsstats.h"
35 #include "tnsclien.h"
36 #include "x86.h"
37
38 //
39 // A structure representing the instance information associated with
40 // a particular device
41 //
42
43 typedef struct _DEVICE_EXTENSION {
44      ULONG  StateVariable;
45 } DEVICE_EXTENSION, *PDEVICE_EXTENSION;
46
47
48 VOID GetSidt(PVOID);
49
50
51 ULONG GTestFlag=10;
52 ULONG _gPrintStats = 0;
53
54
55 extern unsigned char *MyTrap0E;
56
57
58 NTSTATUS
59 TNSClientDrvDispatch(
60      IN PDEVICE_OBJECT DeviceObject,
61      IN PIRP Irp
62      );
63
64 VOID
65 TNSClientDrvUnload(
66      IN PDRIVER_OBJECT DriverObject
67      );
68
69 ULONG PFPrintFlag = FALSE;
70
71 #define TESTTIMES    1000
72
73 //*************************************************************
74 //
75 // Linear congruent pseudorandom number generator for initialization
76 //
77 //*************************************************************
78 unsigned long seed=1;
79
80 //*************************************************************
81 //
82 // Return a pseudorandom number in the interval 0 < n < 32768
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c**        **Page 2 of 9**

```
 83   // This produces the following sequence of pseudorandom numbers:
 84   //   3465, 130, 10982, 1090     (9996 numbers skipped)   23369,
 85   //   2020, 703, 12762, 10828, 16252, 28648, 27041, 23414, 6604
 86   //
 87   //
 88
 89   //
 90   //
 91   unsigned
 92   myrand()
 93   //
 94   // Description:
 95   //     Return a 16-bit random number from a linear congruent pseudorandom
 96   //     number generator in the range 0 <= n < 32768
 97   //
 98   //
 99   {
100       seed = seed*0x015a4e35L + 1;
101       return (seed>>16)&0x7fff;
102   }
103
104   //
105   //
106   unsigned long
107   myrand32()
108   //
109   // Description:
110   //     Return a 32-bit random number from a linear congruent pseudorandom
111   //     number generator in the range 0 <= n < 2^32
112   //
113   //
114   {
115       unsigned long n;
116       n = myrand();
117       n = n << 16;
118       n |= myrand();
119       return n;
120   }
121
122   //
123   //
124   unsigned long
125   myrand32n(unsigned long clipvalue)
126   //
127   // Description:
128   //     Return a 32-bit random number from a linear congruent pseudorandom
129   //     number generator in the range 0 <= n < 2^32
130   //
131   //
132   {
133       unsigned long n;
134       n = myrand();
135       n = n << 16;
136       n |= myrand();
137
138       if (clipvalue == 0)
139           return 1;
140
141       return (n % clipvalue);
142   }
143
144
145   //
146   //
147   unsigned
148   myrandn(
149       unsigned n)          // clip number
150   //
151   // Description:
152   //     Return a clipped random number, this function returns a random
153   //     number in the range 0 <= random < n
154   //
155   //
156   {
157       if (n == 0)
158           return 1;
159
160       return (myrand() % n);
161   }
162
163   //
164   //
```

```
165  //  initialize above linear congruent pseudorandom number generator.
166  //
167  //
168
169
170  //
171  //
172  void
173  mysrand(
174      unsigned newseed)
175  //
176  //  Description:
177  //      Sets the random number generator seed to a new value.
178  //
179  //
180  {
181      seed = newseed;
182  }
183
184  //
185  //
186  unsigned
187  getseed(void)
188  //
189  //  Description:
190  //      Gets the current random number generator seed.
191  //
192  //
193  {
194      return seed;
195  }
196
197
198
199  NTSTATUS
200  DriverEntry(
201      IN PDRIVER_OBJECT  DriverObject,
202      IN PUNICODE_STRING RegistryPath
203      )
204
205
206  Routine Description:
207
208      Installable driver initialization entry point.
209      This entry point is called directly by the I/O system.
210
211
212  Arguments:
213
214
215      RegistryPath - pointer to a unicode string representing the path
216                     to driver-specific key in the registry.
217
218  Return Value:
219
220      STATUS_SUCCESS if successful,
221      STATUS_UNSUCCESSFUL otherwise.
222
223
224  {
225
226      PDEVICE_OBJECT          deviceObject      = NULL;
227      NTSTATUS                ntStatus;
228      WCHAR                   deviceNameBuffer[] = L"\\Device\\TNSCLIEN";
229      UNICODE_STRING          deviceNameUnicodeString;
230      PDEVICE_EXTENSION       deviceExtension;
231      WCHAR                   deviceLinkBuffer[] = L"\\DosDevices\\TNSCLIEN";
232      UNICODE_STRING          deviceLinkUnicodeString;
233
234      IDTRRegisterContents    IDTRContents;
235      PIDTREntry              pIdtrEntry;
236      int i;
237      ULONG                   NewAddress;
238      LARGE_INTEGER           tsc1, tsc2, tscdiff;
239      PHYSICAL_ADDRESS        pAddr;
240      PVOID                   pBuffer;
241      PVOID                   pMapBuffer;
242
243      //
244      //  Refuse to load if
245      //
246      //  Ensure there are no resources (KdPrint(ResourceUsage));
```

File: D:\nt4DDK\src\timean\tnsclien\tnsclien.c                        Page 4 of 9

```
247     //
248     //   2. Attempt to locate the device(s) it supports
249     // OR we've claimed our resources & found our h/w so create
250     // a device and initialize stuff.
251     //
252
253     RtlInitUnicodeString(&deviceNameUnicodeString,
254                          deviceNameBuffer);
255
256
257
258     //
259     // Create an EXCLUSIVE device. One only 1 thread at a time can send
260     // i/o requests.
261     //
262
263     ntStatus = IoCreateDevice (DriverObject,
264                                sizeof (DEVICE_EXTENSION),
265                                &deviceNameUnicodeString,
266                                FILE_DEVICE_TNSCLIENT,
267                                0,
268                                TRUE,
269                                &deviceObject
270                                );
271
272     if (NT_SUCCESS(ntStatus)) {
273         deviceExtension = (PDEVICE_EXTENSION) deviceObject->DeviceExtension;
274
275
276
277         //
278         // Set up synchronization objects, state, etc.
279         //
280
281
282
283         //
284         // Create a symbolic link that Win32 apps can specify to gain access
285         // to this driver/device
286         //
287
288         RtlInitUnicodeString (&deviceLinkUnicodeString, deviceLinkBuffer);
289
290         ntStatus = IoCreateSymbolicLink (&deviceLinkUnicodeString, &deviceNameUnicodeString);
291
292
293         if (!NT_SUCCESS(ntStatus)) {
294             _asm int 3
295         }
296
297
298
299         //
300         // Create dispatch points for device control, create, close.
301         //
302
303         DriverObject->MajorFunction[IRP_MJ_CREATE]          =
304         DriverObject->MajorFunction[IRP_MJ_CLOSE]           =
305         DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]  = TNSClientDrvDispatch;
306         DriverObject->DriverUnload                          = TNSClientDrvUnload;
307     }
308
309
310     if (!NT_SUCCESS(ntStatus)) {
311         //
312         // Something went wrong, so clean up whatever resources we ...
313         //
314
315         if (deviceObject)
316             IoDeleteDevice (deviceObject);
317     }
318
319     return ntStatus;
320 }
321
322
323 ULONG
324 _declspec(dllimport)
325 __TNS_READ_REGISTER_ULONG(
326     PVOID   DeviceContext,
327     PULONG  Register);
328
```

```
329
330 ULONG
331 _declspec(dllimport)
332 _TNS_WRITE_REGISTER_ULONG(
333     PVOID    DeviceContext,
334     PULONG   Register,
335     ULONG    RegisterData);
336
337
338 ULONG
339 _declspec(dllimport)
340 _TNS_GET_SMN_STATISTICS(
341     IN       PVOID      DeviceHandle,
342     IN OUT   PSTATISTICS pStatistics,
343     IN OUT   PULONG     pStatsStructSize,
344     IN OUT   pMPSTATS   pMpStats,
345     IN OUT   PULONG     pMpStatsSize);
346
347 ULONG
348 _declspec(dllimport)
349 _TNS_GET_NODE_STATISTICS(
350     IN       PVOID      DeviceHandle,
351     IN OUT   PSTATISTICS pStatistics,
352     IN OUT   PULONG     pStatsStructSize,
353     IN OUT   pMPSTATS   pMpStats,
354     IN OUT   PULONG     pMpStatsSize);
355
356
357
358 ULONG
359 _declspec(dllimport)
360 _TNS_CLEAR_NODE_STATISTICS(
361     IN       PVOID      DeviceHandle);
362
363 ULONG
364 _declspec(dllimport)
365 _TNS_CLEAR_SMN_STATISTICS(
366     IN       PVOID      DeviceHandle);
367
368 ULONG
369 _declspec(dllimport)
370 _TNS_GET_SMN_INFORMATION(
371     IN       PVOID      DeviceHandle,
372     IN OUT   unsigned char *pMacAddress,
373     IN OUT   unsigned char *pNodeName,
374     IN OUT   unsigned long *pSharedMemorySize);
375
376 ULONG
377 _declspec(dllimport)
378 _TNS_GET_SMN_TABLE_INFO(
379     IN       PVOID      DeviceHandle,
380     IN OUT   pSMNTableInfo pSMNInfo);
381
382 ULONG
383 _declspec(dllimport)
384 _TNS_GET_SMN_STATISTICS_BY_NODEID(
385     IN       PVOID      DeviceHandle,
386     IN       ULONG      NodeID,
387     IN OUT   PSTATISTICS pStatistics,
388     IN OUT   PULONG     pStatsStructSize,
389     IN OUT   pMPSTATS   pMpStats,
390     IN OUT   PULONG     pMpStatsSize);
391
392 ULONG
393 _declspec(dllimport)
394 _TNS_GET_NODE_INFORMATION(
395     IN       PVOID      DeviceHandle,
396     IN OUT   unsigned char *pMacAddress,
397     IN OUT   unsigned char *pNodeName,
398     IN OUT   unsigned int  *pNodeID);
399
400 NTSTATUS
401 TNSClientDrvDispatch(
402     IN PDEVICE_OBJECT DeviceObject,
403     IN PIRP           Irp
404     )
405
406
407
408
409
410
```

File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c                   **Page    of 9**

```
411   
412   
413   
414   
415   
416   
417   
418   
419   
420   
421  (
422   
423      PIO_STACK_LOCATION   irpStack;
424      PDEVICE_EXTENSION    deviceExtension;
425      pIO_DRIVER_PACKET    ioBuffer;
426      ULONG                inputBufferLength;
427      ULONG                outputBufferLength;
428      ULONG                ioControlCode;
429      NTSTATUS             ntStatus;
430      int i;
431   
432      ULONG                ReturnCode;
433   
434   
435      Irp->IoStatus.Status      = STATUS_SUCCESS;
436      Irp->IoStatus.Information = 0;
437   
438   
439   
440   
441   
442   
443   
444      irpStack = IoGetCurrentIrpStackLocation (Irp);
445   
446   
447   
448   
449   
450   
451   
452      deviceExtension = DeviceObject->DeviceExtension;
453   
454   
455   
456   
457   
458   
459   
460      ioBuffer          = (pIO_DRIVER_PACKET)Irp->AssociatedIrp.SystemBuffer;
461      inputBufferLength = irpStack->Parameters.DeviceIoControl.InputBufferLength;
462      outputBufferLength = irpStack->Parameters.DeviceIoControl.OutputBufferLength;
463   
464   
465   
466      switch (irpStack->MajorFunction) {
467          case IRP_MJ_CREATE:
468   
469              break;
470   
471          case IRP_MJ_CLOSE:
472   
473              break;
474   
475          case IRP_MJ_DEVICE_CONTROL:
476   
477              ioControlCode = irpStack->Parameters.DeviceIoControl.IoControlCode;
478   
479              switch (ioControlCode) {
480   
481                  case IOCTL_TNSCLIENT_GET_NODE_INFO: {
482                      ULONG StatsLen, mpStatsLen;
483   
484                      mpStatsLen = sizeof(MPSTATS);
485                      StatsLen = sizeof(STATISTICS);
486   
487                      __TNS_GET_SMN_STATISTICS_BY_NODEID(
488                          NULL,
489                          ioBuffer->TeamNodeID,
490                          &ioBuffer->Stats,
491                          &StatsLen,
492                          &ioBuffer->MpStats,
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c**             **Page 7 of 9**

```
493                         &mpStatsLen);
494
495                     Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
496                     break;
497             }
498
499             case IOCTL_TNSCLIENT_GET_SMN_TABLE_INFO: {
500                 __TNS_GET_SMN_TABLE_INFO(
501                     NULL,
502                     ioBuffer->SMNInfo);
503
504                     Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
505                     break;
506             }
507             case IOCTL_TNSCLIENT_GET_SMN_INFO: {
508                 __TNS_GET_SMN_INFORMATION(
509                     NULL,
510                     ioBuffer->MacAddress,
511                     ioBuffer->ComputerName,
512                     &ioBuffer->TNSSharedMemorySize);
513
514                     Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
515                     break;
516             }
517
518             case IOCTL_TNSCLIENT_CLEAR_STATS: {
519                 __TNS_CLEAR_NODE_STATISTICS(
520                     NULL);
521                 __TNS_CLEAR_SMN_STATISTICS(
522                     NULL);
523                     Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
524                     break;
525             }
526
527             case IOCTL_TNSCLIENT_GET_LOCAL_INFO: {
528                 __TNS_GET_NODE_INFORMATION(
529                     NULL,
530                     ioBuffer->MacAddress,
531                     ioBuffer->ComputerName,
532                     &ioBuffer->TeamNodeID);
533                     Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
534                     break;
535             }
536
537             case IOCTL_TNSCLIENT_DOTEST: {
538                 int i;
539                 unsigned long randdata;
540                 unsigned long randaddress;
541                 unsigned long returndata;
542
543                 if (ioBuffer->MaxNumWrites) {
544                     for (i=0; i<ioBuffer->MaxNumWrites; i++) {
545                         randdata = myrand32();
546                         randaddress = myrand32n(ioBuffer->TNSSharedMemorySize);
547                         __TNS_WRITE_REGISTER_ULONG(NULL, (PULONG)randaddress, randdata);
548                     }
549                 }
550
551                 if (ioBuffer->MaxNumReads) {
552                     for (i=0; i<ioBuffer->MaxNumReads; i++) {
553                         randaddress = myrand32n(ioBuffer->TNSSharedMemorySize);
554                         returndata = __TNS_READ_REGISTER_ULONG(NULL, (PULONG) randaddress);
555                     }
556                 }
557
558                 if (ioBuffer->MaxNumReadWrites) {
559                     for (i=0; i<ioBuffer->MaxNumReadWrites; i++) {
560                         randdata = myrand32();
561                         randaddress = myrand32n(ioBuffer->TNSSharedMemorySize);
562
563                         __TNS_WRITE_REGISTER_ULONG(NULL, (PULONG)randaddress, randdata);
564                         returndata = __TNS_READ_REGISTER_ULONG(NULL, (PULONG)randaddress);
565                         if (randdata != returndata) {
566                             DbgPrint("randdata != returndata, randdata => %x, returndata => %x\n", ra
-2 nddata, returndata);
567                             break;
568                         }
569                     }
570                 }
571
572                 Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
573                 break;
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c**                        **Pag  8 of 9**

```
574                    )
575
576                    case IOCTL_TNSCLIENT_GET_LOCAL_STATS: (
577                        ULONG StatsLen, mpStatsLen;
578
579                        mpStatsLen = sizeof(MPSTATS);
580                        StatsLen = sizeof(STATISTICS);
581
582                        __TNS_GET_NODE_STATISTICS(
583                            NULL,
584                            &ioBuffer->Stats,
585                            &StatsLen,
586                            &ioBuffer->MpStats,
587                            &mpStatsLen);
588
589                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
590                        break;
591                    )
592
593                    case IOCTL_TNSCLIENT_GET_SMN_STATS: (
594                        ULONG StatsLen, mpStatsLen;
595
596                        mpStatsLen = sizeof(MPSTATS);
597                        StatsLen = sizeof(STATISTICS);
598
599                        __TNS_GET_SMN_STATISTICS(
600                            NULL,
601                            &ioBuffer->Stats,
602                            &StatsLen,
603                            &ioBuffer->MpStats,
604                            &mpStatsLen);
605
606                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
607                        break;
608                    )
609
610                    default:
611
612                        Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
613
614                        break;
615
616                }
617            break;
618    }
619
620
621        ▨
622        ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
623        ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
624        ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
625        ▨
626
627    ntStatus = Irp->IoStatus.Status;
628
629    IoCompleteRequest (Irp,
630                       IO_NO_INCREMENT
631                       );
632
633
634        ▨
635        ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
636        ▨
637
638    return ntStatus;
639 }
640
641
642
643 VOID
644 TNSClientDrvUnload(
645     IN PDRIVER_OBJECT DriverObject
646     )
647 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
648 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
649 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
650 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
651 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
652 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
653 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
654 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
655 ░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c**                                              **Page 9 of 9**

```
656
657   Return Value:
658
659
660   --*/
661   {
662       WCHAR                deviceLinkBuffer[]  = L"\\DosDevices\\TNSCLIEN";
663       UNICODE_STRING       deviceLinkUnicodeString;
664       IDTRRegisterContents IDTRContents;
665       PIDTREntry           pIdtrEntry;
666
667       //
668       // Delete the symbolic link
669       //
670
671       RtlInitUnicodeString (&deviceLinkUnicodeString, deviceLinkBuffer);
672
673       IoDeleteSymbolicLink (&deviceLinkUnicodeString);
674
675       //
676       // Delete the device object
677       //
678
679       IoDeleteDevice (DriverObject->DeviceObject);
680   }
```

## CLAIMS

What is claimed is:

5

1.      A method, comprising:

passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including:

10                      receiving said set of interconnect fabric data with said shim layer,

classifying said set of interconnect fabric data with said shim layer, and

handling said set of interconnect fabric data with said shim layer

15      as a function of a transport application program interface with which said set of interconnect fabric data is associated.


2.      The method of claim 1, wherein said set of interconnect fabric data includes a packet.

20

3.      The method of claim 1, wherein classifying said set of interconnect fabric data includes classifying said set of interconnect fabric data as a function of said transport application program interface.


25      4.      The method of claim 1, wherein said set of interconnect fabric data is received and then classified and then passed.


5.      The method of claim 1, wherein passing includes transforming said set of interconnect fabric data.

30

6.      The method of claim 1, further comprising monitoring passage of said set of interconnect fabric data with a heartbeat function to expedite recovery in the event of an error.

7.      The method of claim 1, further comprising monitoring passage of said set of interconnect fabric data with sense interrupt indications to expidite recovery in the event of an error.

5

8.      A method, comprising:

passing a set of network data through a shim layer that is interposed between a network interface layer and a protocol layer including:

receiving said set of network data with said shim layer,

10                      classifying said set of network data with said shim layer, and

handling said set of network data with said shim layer as a function of a transport application program interface with which said set of network data is associated.

15      9.      The method of claim 8, wherein said set of network data includes a packet.

10.     The method of claim 8, wherein classifying said set of network data includes classifying said set of network data as a function of said transport

20      application program interface.

11.     The method of claim 8, wherein said set of network data is received and then classified and then handled.

25      12.     The method of claim 8, wherein passing includes transforming said said of network data.

13.     The method of claim 8, further comprising monitoring passage of said set of network data with a heartbeat function to expedite recovery in the event of

30      an error.

14.     The method of claim 8, further comprising monitoring passage of said set of network data with sense interrupt indications to expedite recovery in the

event of an error.

15.     The method of claim 8, wherein said shim hosts network middleware to handle at least one function selected from the group consisting of transmitting packets, obtaining information on local and remote multi-computer nodes, setting up packet receive sinks and controlling a protocol.

16.     An apparatus, comprising:
        a shared memory unit;
        a first system coupled to said shared memory unit; and
        a second system coupled to said shared memory unit,
        wherein a data set transfered between said shared memory unit and at least one member selected from the group consisiting of said first system and said second system is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated.

17.     A computer system comprising the apparatus of claim 16.

18.     The apparatus of claim 16, wherein the shim is interposed between said network device/driver and said protocol layer, and said at least one member includes a network interface card.

19.     The apparatus of claim 18, wherein the network interface card provides a heartbeat function to facilitate error recovery.

20.     The apparatus of claim 18, wherein the network interface card provides programable packet type identification.

21.     The apparatus of claim 18, wherein the network interface card provides media sense interrupt indications to facilitate error recovery.

22.     The apparatus of claim 16, wherein the shim is interposed between said interconnect fabric interface and said protocol layer.

5       23.     The apparatus of claim 22, wherein said at least one member provides a heartbeat function to facilitate error recovery.

24.     The apparatus of claim 22, wherein said at least one member provides programable packet type identification.

10

25.     The apparatus of claim 22, wherein said at least one member provides media sense interrupt indications to facilitate error recovery.

26.     An apparatus, comprising:

15              a switch;

                a first system coupled to said switch; and

                a second system node coupled to said switch,

                wherein a data set transfered from said first system to said second system through said switch is received by a shim that is interposed between

20      either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated.

25      27.     A computer system comprising the apparatus of claim 26.

28.     The apparatus of claim 26, wherein the shim is interposed between said network device/driver and said protocol layer, and said at least one member includes a network interface card.

30

29.     The apparatus of claim 28, wherein the network interface card provides a heartbeat function to facilitate error recovery.

30.    The apparatus of claim 28, wherein the network interface card provides programable packet type identification.

31.    The apparatus of claim 28, wherein the network interface card provides
5      media sense interrupt indications to facilitate error recovery.

32.    The apparatus of claim 26, wherein the shim is interposed between said interconnect fabric interface and said protocol layer.

10     33.    The apparatus of claim 32, wherein said at least one member provides a heartbeat function to facilitate error recovery.

34.    The apparatus of claim 32, wherein said at least one member provides programable packet type identification.

15

35.    The apparatus of claim 32, wherein said at least one member provides media sense interrupt indications to facilitate error recovery.

36.    An electronic media, comprising: a computer program adapted to pass a
20     set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including:

receiving said set of interconnect fabric data with said shim

layer,

classifying said set of interconnect fabric data with said shim
25     layer, and

handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated.

30     37.    A computer program comprising computer program means adapted to perform the steps of passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including:

receiving said set of interconnect fabric data with said shim

layer,

classifying said set of interconnect fabric data with said shim

layer, and

5          handling said set of interconnect fabric data with said shim layer

as a function of a transport application program interface with which said set of

interconnect fabric data is associated when said computer program is run on a

computer.

10        38.      A computer program as claimed in claim 37, embodied on a computer-

readable medium.

39.      An electronic media, comprising: a computer program adapted to pass a

set of network data through a shim layer that is interposed between a network

15        interface layer and a protocol layer including:

receiving said set of network data with said shim layer,

classifying said set of network data with said shim layer, and

handling said set of network data with said shim layer as a

function of a transport application program interface with which said set of

20        network data is associated.

40.      A computer program comprising computer program means adapted to

perform the steps of passing a set of network data through a shim layer that is

interposed between a network interface layer and a protocol layer including:

25        receiving said set of network data with said shim layer,

classifying said set of network data with said shim layer, and

handling said set of network data with said shim layer as a

function of a transport application program interface with which said set of

network data is associated when said computer program is run on a computer.

30

41.      A computer program as claimed in claim 40, embodied on a computer-
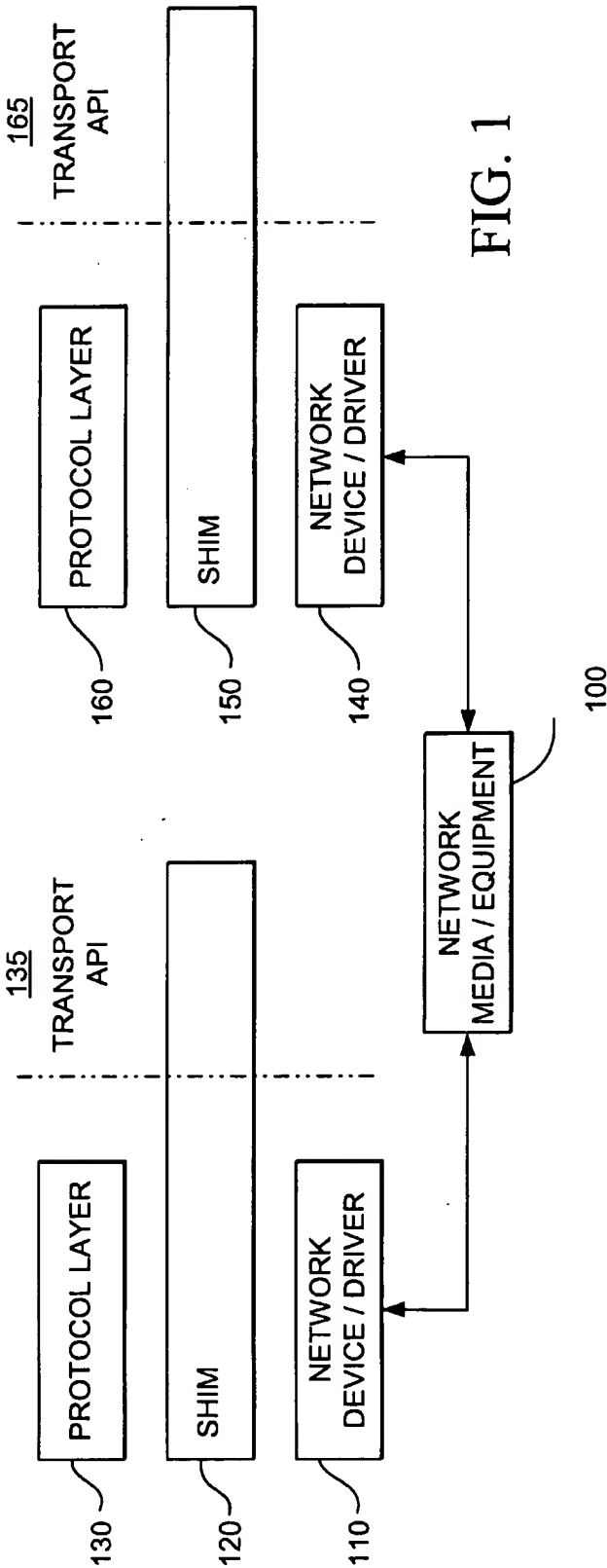
readable medium.
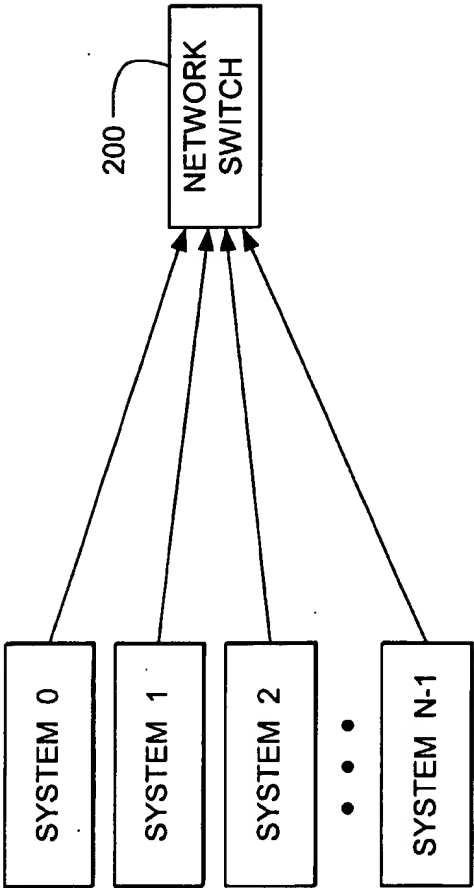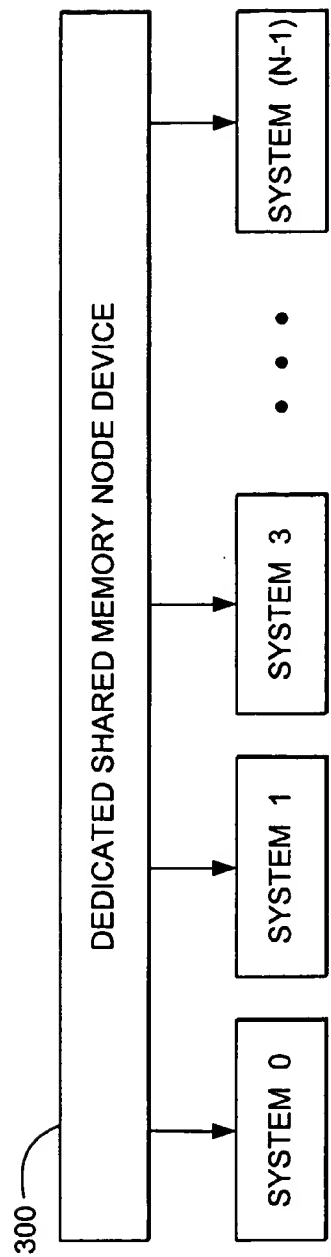
1/2



FIG. 1



FIG. 2

FIG. 3



FIG. 4